数据挖掘十大算法详解

wizardforcel

Published with GitBook

目錄

数据挖掘十大算法详解

数据挖掘学习笔记--决策树C4.5

> 来源：http://blog.csdn.net/u011067360/article/details/21861989

在网上和教材上也看了有很多数据挖掘方面的很多知识，自己也学习很多，就准备把自己学习和别人分享的结合去总结下，以备以后自己回头看，看别人总还是比不上自己写点，及时有些不懂或者是没有必要。

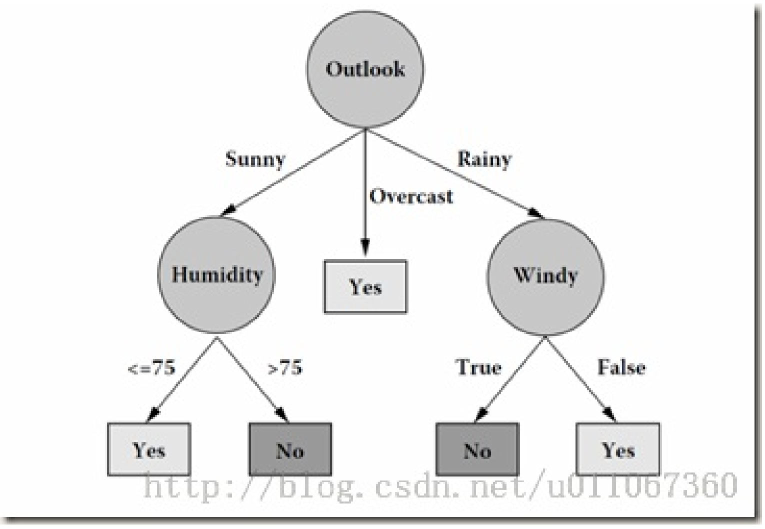
定义：分类树（决策树）是一种十分常用的分类方法。他是一种监管学习，所谓监管学习说白了很简单，就是给定一堆样本，每个样本都有一组属性和一个类别，这些类别是事先确定的，那么通过学习得到一个分类器，这个分类器能够对新出现的对象给出正确的分类。这样的机器学习就被称之为监督学习。分类本质上就是一个map的过程。C4.5分类树就是决策树算法中最流行的一种。

算法简介：

```
Function C4.5(R:包含连续属性的无类别属性集合,C:类别属性,S:训练集)
/*返回一棵决策树*/
Begin
   If S为空,返回一个值为Failure的单个节点;
   If S是由相同类别属性值的记录组成,
      返回一个带有该值的单个节点;
   If R为空,则返回一个单节点,其值为在S的记录中找出的频率最高的类别属性值;
   [注意未出现错误则意味着是不适合分类的记录]；
  For 所有的属性R(Ri) Do
         If 属性Ri为连续属性，则
      Begin
            将Ri的最小值赋给A1：
         将Rm的最大值赋给Am；/*m值手工设置*/
            For j From 2 To m-1 Do Aj=A1+j*(A1Am)/m;
            将Ri点的基于{< =Aj,>Aj}的最大信息增益属性(Ri,S)赋给A；
      End；
   将R中属性之间具有最大信息增益的属性(D,S)赋给D;
    将属性D的值赋给{dj/j=1,2...m}；
   将分别由对应于D的值为dj的记录组成的S的子集赋给{sj/j=1,2...m};
    返回一棵树，其根标记为D;树枝标记为d1,d2...dm;
    再分别构造以下树:
    C4.5(R-{D},C,S1),C4.5(R-{D},C,S2)...C4.5(R-{D},C,Sm);
 End C4.5
```

```
Function C4.5(R:包含连续属性的无类别属性集合,C:类别属性,S:训练集)
/*返回一棵决策树*/
Begin
   If S为空,返回一个值为Failure的单个节点;
   If S是由相同类别属性值的记录组成,
      返回一个带有该值的单个节点;
   If R为空,则返回一个单节点,其值为在S的记录中找出的频率最高的类别属性值;
   [注意未出现错误则意味着是不适合分类的记录];
  For 所有的属性R(Ri) Do
        If 属性Ri为连续属性，则
     Begin
           将Ri的最小值赋给A1：
        将Rm的最大值赋给Am；/*m值手工设置*/
           For j From 2 To m-1 Do Aj=A1+j*(A1Am)/m;
           将Ri点的基于{< =Aj,>Aj}的最大信息增益属性(Ri,S)赋给A；
     End；
  将R中属性之间具有最大信息增益的属性(D,S)赋给D;
   将属性D的值赋给{dj/j=1,2...m}；
  将分别由对应于D的值为dj的记录组成的S的子集赋给{sj/j=1,2...m};
   返回一棵树，其根标记为D;树枝标记为d1,d2...dm;
   再分别构造以下树:
   C4.5(R-{D},C,S1),C4.5(R-{D},C,S2)...C4.5(R-{D},C,Sm);
End C4.5
```

该算法的框架表述还是比较清晰的，从根节点开始不断得分治，递归，生长，直至得到最后的结果。根节点代表整个训练样本集,通过在每个节点对某个属性的测试验证,算法递归得将数据集分成更小的数据集.某一节点对应的子树对应着原数据集中满足某一属性测试的部分数据集.这个递归过程一直进行下去,直到某一节点对应的子树对应的数据集都属于同一个类为止,例如数据集对应得到的决策树如下:



分类树中的测试是怎样的？

分类树中的测试是针对某一个样本属性进行的测试。样本的属性有两种，一种是离散变量，一种是连续变量。对于离散变量，这很简单，离散变量对应着多个值，每个值就对应着测试的一个分支，测试就是验证样本对应的属性值对应哪个分支。这样数据集就会被分成几个小组。对于连续变量，所有连续变量的测试分支都是2条，其测试分支分别对应着{<=A?,>a?}(a就是对应的阀值)。

如何选择测试？

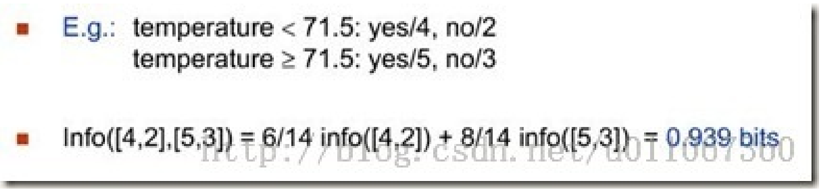
分类树中每个节点对应着测试，但是这些测试是如何来选择呢？C4.5根据信息论标准来选择测试，比如增益（在信息论中，熵对应着某一分布的信息量，其值同时也对应着要完全无损表示该分布所需要的最小的比特数，本质上熵对应着不确定性，可能的变化的丰富程度。所谓增益，就是指在应用了某一测试之后，其对应的可能性丰富程度下降，不确定性减小，这个减小的幅度就是增益，其实质上对应着分类带来的好处）或者增益比（这个指标实际上就等于增益/熵，之所以采用这个指标是为了克服采用增益作为衡量标准的缺点，采用增益作为衡量标准会导致分类树倾向于优先选择那些具有比较多的分支的测试，这种倾向需要被抑制）。算法在进行Tree-Growth时，总是“贪婪得”选择那些信息论标准最高的那些测试。

如何选择连续变量的阈值？

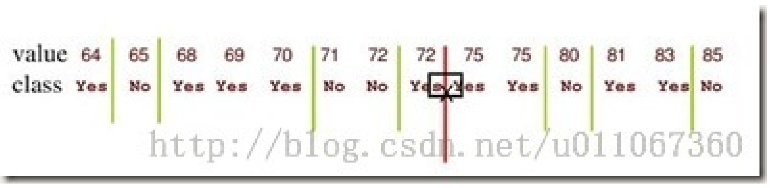
在《分类树中的测试是怎样的？》中提到连续变量的分支的阈值点为a，这阈值如何确定呢？很简单，把需要处理的样本（对应根节点）或样本子集（对应子树）按照连续变量的大小从小到大进行排序，假设该属性对应的不同的属性值一共有N个，那么总共有N-1个可能的候选分割阈值点，每个候选的分割阈值点的值为上述排序后的属性值链表中两两前后连续元素的中点，那么我们的任务就是从这个N-1个候选分割阈值点中选出一个，使得前面提到的信息论标准最大。举个例子，对play数据集，我们来处理温度属性，来选择合适的阈值。首先按照温度大小对对应样本进行排序如下：



那么可以看到有13个可能的候选阈值点，比如middle[64,65], middle[65,68] ….,middle[83,85]。那么最优的阈值该选多少呢？应该是middle[71,72]，如上图中红线所示。为什么呢？如下计算：



通过上述计算方式，0.939是最大的，因此测试的增益是最小的。（测试的增益和测试后的熵是成反比的，这个从后面的公式可以很清楚的看到）。根据上面的描述，我们需要对每个候选分割阈值进行增益或熵的计算才能得到最优的阈值，我们需要算N-1次增益或熵（对应温度这个变量而言就是13次计算）。能否有所改进呢？少算几次，加快速度。答案是可以该进，如下图：



该图中的绿线代表可能的最优分割阈值点，根据信息论知识，像middle[72,75]（红线所示）这个分割点，72,75属于同一个类，这样的分割点是不可能有信息增益的。（把同一个类分成了不同的类，这样的阈值点显然不会有信息增益，因为这样的分类没能帮上忙，减少可能性）

哪个属性是最佳的分类属性？

信息论标准有两种，一种是增益，一种是增益比。

首先来看看增益Gain的计算

为了精确地定义信息增益，我们先定义信息论中广泛使用的一个度量标准，称为熵（entropy），它刻画了任意样例集的纯度（purity）。给定包含关于某个目标概念的正反样例的样例集S，那么S相对这个布尔型分类的熵为：

$$Entropy(S) \equiv -p_{\oplus}\log_2 p_{\oplus} - p_{\ominus}\log_2 p_{\ominus}$$

上述公式中，p+代表正样例，比如在本文开头第二个例子中p+则意味着去打羽毛球，而p-则代表反样例，不去打球。

举例来说，假设S是一个关于布尔概念的有14个样例的集合，它包括9个正例和5个反例（我们采用记号[9+，5-]来概括这样的数据样例），那么S相对于这个布尔样例的熵为：

```
Entropy（[9+，5-]）=-（9/14）log2（9/14）-（5/14）log2（5/14）=0.940。
```

上述公式的值为0.940。它的信息论含义就是我要想把Play？这个信息传递给别人话，平均来讲我至少需要0.940个bit来传递这个信息。C4.5的目标就是经过分类来减小这个熵。那么我们来依次考虑各个属性测试，通过某一属性测试我们将样本分成了几个子集，这使得样本逐渐变得有序，那么熵肯定变小了。这个熵的减小量就是我们选择属性测试的依据。

信息增益Gain(S,A)定义

已经有了熵作为衡量训练样例集合纯度的标准，现在可以定义属性分类训练数据的效力的度量标准。这个标准被称为“信息增益（information gain）”。简单的说，一个属性的信息增益就是由于使用这个属性分割样例而导致的期望熵降低(或者说，样本按照某属性划分时造成熵减少的期望)。更精确地讲，一个属性A相对样例集合S的信息增益Gain(S,A)被定义为：

$$Gain(S,A) = Entropy(S) - \sum_{v \in V(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

其中 Values(A)是属性A所有可能值的集合，是S中属性A的值为v的子集。换句话来讲，Gain(S,A)是由于给定属性A的值而得到的关于目标函数值的信息。当对S的一个任意成员的目标值编码时，Gain(S,A)的值是在知道属性A的值后可以节省的二进制位数。

它的实质是把数据集D根据某一属性测试分成v个子集，这使得数据集S变得有序，使得数据集S的熵变小了。分组后的熵其实就是各个子集的熵的权重和。通过计算我们得到Gain(Outlook)=0.940-0.694=0.246，Gain(Windy)=0.940-0.892=0.048….

可以得到第一个测试属性是Outlook。需要注意的是，属性测试是从数据集中包含的所有属性组成的候选属性中选择出来的。对于所在节点到根节点的路径上所包含的属性（我们称之为继承属性），其实根据公式很容易得到他们的熵增益是0，因此这些继承属性完全不必考虑，可以从候选属性中剔除这些属性。

到这里似乎一切都很完美，增益这个指标非常好，但是其实增益这个指标有一个缺点。我们来考虑play数据集中的Day这个属性（我们假设它是一个真属性，实际上很可能大家不会把他当做属性），Day有14个不同的值，那么Day的属性测试节点就会有14个分支，很显然每个分支其实都覆盖了一个“纯”数据集（所谓“纯”，指的就是所覆盖的数据集都属于同一个类），那么其熵增益显然就是最大的，那么Day就默认得作为第一个属性。之所以出现这样的情况，是因为增益这个指标天然得偏向于选择那些分支比较多的属性，也就是那些具有的值比较多的那些属性。这种偏向性使我们希望克服的，我们希望公正地评价所有的属性。因此又一个指标被提出来了Gain Ratio-增益比。

C4.5算法之信息增益率

增益比率度量是用前面的增益度量Gain(S，A)和分裂信息度量SplitInformation(S，A)来共同定义的，如下所示：

$$GainRatio(S,A) \equiv \frac{Gain(S,A)}{SplitInformation(S,A)}$$

其中，分裂信息度量被定义为(分裂信息用来衡量属性分裂数据的广度和均匀)：

$$SplitInformation(S,A) \equiv -\sum_{i=1}^{c}\frac{|S_i|}{|S|}\log_2\frac{|S_i|}{|S|}$$

其中S1到Sc是c个值的属性A分割S而形成的c个样例子集。分裂信息实际上就是S关于属性A的各值的熵。这与我们前面对熵的使用不同，在那里我们只考虑S关于学习到的树要预测的目标属性的值的熵。

通过计算我们很容易得到GainRatio(Outlook)=0.246/1.577=0.156。增益比实际上就是对增益进行了归一化，这样就避免了指标偏向分支多的属性的倾向。

决策树能够帮助我们对新出现的样本进行分类，但还有一些问题它不能很好得解决。比如我们想知道对于最终的分类，哪个属性的贡献更大？能否用一种比较简洁的规则来区分样本属于哪个类？等等。

C4.5算法的改进：

用信息增益率来选择属性。

在树构造过程中进行剪枝，在构造决策树的时候，那些挂着几个元素的节点，不考虑最好，不然容易导致overfitting。

对非离散数据也能处理。

能够对不完整数据进行处理。

决策树的剪枝

决策树为什么要剪枝？原因就是避免决策树“过拟合”样本。前面的算法生成的决策树非常的详细而庞大，每个属性都被详细地加以考虑，决策树的树叶节点所覆盖的训练样本都是“纯”的。因此用这个决策树来对训练样本进行分类的话，你会发现对于训练样本而言，这个树表现堪称完美，它可以100%完美正确得对训练样本集中的样本进行分类（因为决策树本身就是100%完美拟合训练样本的产物）。但是，这会带来一个问题，如果训练样本中包含了一些错误，按照前面的算法，这些错误也会100%一点不留得被决策树学习了，这就是“过拟合”。C4.5的缔造者昆兰教授很早就发现了这个问题，他作过一个试验，在某一个数据集中，过拟合的决策树的错误率比一个经过简化了的决策树的错误率要高。那么现在的问题就来了，如何在原生的过拟合决策树的基础上，通过剪枝生成一个简化了的决策树？

1、第一种方法，也是最简单的方法，称之为基于误判的剪枝。这个思路很直接，完全的决策树不是过度拟合么，我再搞一个测试数据集来纠正它。对于完全决策树中的每一个非叶子节点的子树，我们尝试着把它替换成一个叶子节点，该叶子节点的类别我们用子树所覆盖训练样本中存在最多的那个类来代替，这样就产生了一个简化决策树，然后比较这两个决策树在测试数据集中的表现，如果简化决策树在测试数据集中的错误比较少，并且该子树里面没有包含另外一个具有类似特性的子树（所谓类似的特性，指的就是把子树替换成叶子节点后，其测试数据集误判率降低的特性），那么该子树就可以替换成叶子节点。该算法以bottom-up的方式遍历所有的子树，直至没有任何子树可以替换使得测试数据集的表现得以改进时，算法就可以终止。

2、第一种方法很直接，但是需要一个额外的测试数据集，能不能不要这个额外的数据集呢？为了解决这个问题，于是就提出了悲观剪枝。该方法剪枝的依据是训练样本集中的样本误判率。我们知道一颗分类树的每个节点都覆盖了一个样本集，根据算法这些被覆盖的样本集往往都有一定的误判率，因为如果节点覆盖的样本集的个数小于一定的阈值，那么这个节点就会变成叶子节点，所以叶子节点会有一定的误判率。而每个节点都会包含至少一个的叶子节点，所以每个节点也都会有一定的误判率。悲观剪枝就是递归得估算每个内部节点所覆盖样本节点的误判率。剪枝后该内部节点会变成一个叶子节点，该叶子节点的类别为原内部节点的最优叶子节点所决定。然后比较剪枝前后该节点的错误率来决定是否进行剪枝。该方法和前面提到的第一种方法思路是一致的，不同之处在于如何估计剪枝前分类树内部节点的错误率。

连续值属性的改进

相对于那些离散值属性，分类树算法倾向于选择那些连续值属性，因为连续值属性会有更多的分支，熵增益也最大。算法需要克服这种倾向，我们利用增益率来克服这种倾向。增益率也可以用来克服连续值属性倾向。增益率作为选择属性的依据克服连续值属性倾向，这是没有问题的。但是如果利用增益率来选择连续值属性的分界点，会导致一些副作用。分界点将样本分成两个部分，这两个部分的样本个数之比也会影响增益率。根据增益率公式，我们可以发现，当分界点能够把样本分成数量相等的两个子集时（我们称此时的分界点为等分分界点），增益率的抑制会被最大化，因此等分分界点被过分抑制了。子集样本个数能够影响分界点，显然不合理。因此在决定分界点是还是采用增益这个指标，而选择属性的时候才使用增益率这个指标。这个改进能够很好得抑制连续值属性的倾向。当然还有其它方法也可以抑制这种倾向，比如MDL。

处理缺失属性

如果有些训练样本或者待分类样本缺失了一些属性值，那么该如何处理？要解决这个问题，需要考虑3个问题：

i)当开始决定选择哪个属性用来进行分支时，如果有些训练样本缺失了某些属性值时该怎么办？

ii)一个属性已被选择，那么在决定分支的时候如果有些样本缺失了该属性该如何处理？

iii)当决策树已经生成，但待分类的样本缺失了某些属性，这些属性该如何处理？针对这三个问题，昆兰提出了一系列解决的思路和方法。

对于问题i),计算属性a的增益或者增益率时，如果有些样本没有属性a，那么可以有这么几种处理方式：

(1)忽略这些缺失属性a的样本。

(2)给缺失属性a的样本赋予属性a一个均值或者最常用的的值。

(3)计算增益或者增益率时根据缺失属性样本个数所占的比率对增益/增益率进行相应的“打折”。

(4)根据其他未知的属性想办法把这些样本缺失的属性补全。 对于问题ii)，当属性a已经被选择，该对样本进行分支的时候，如果有些样本缺失了属性a,那么：

(1)忽略这些样本。

(2)把这些样本的属性a赋予一个均值或者最常出现的值，然后再对他们进行处理。

(3)把这些属性缺失样本，按照具有属性a的样本被划分成的子集样本个数的相对比率，分配到各个子集中去。至于哪些缺失的样本被划分到子集1，哪些被划分到子集2，这个没有一定的准则，可以随机而动。(A)把属性缺失样本分配给所有的子集，也就是说每个子集都有这些属性缺失样本。

(3)单独为属性缺失的样本划分一个分支子集。

(4)对于缺失属性a的样本，尝试着根据其他属性给他分配一个属性a的值，然后继续处理将其划分到相应的子集。 对于问题iii)，对于一个缺失属性a的待分类样本，有这么几种选择：

(1)如果有单独的确实分支，依据此分支。

(2)把待分类的样本的属性a值分配一个最常出现的a的属性值，然后进行分支预测。

(3)根据其他属性为该待分类样本填充一个属性a值，然后进行分支处理。

(4)在决策树中属性a节点的分支上，遍历属性a节点的所有分支，探索可能所有的分类结果，然后把这些分类结果结合起来一起考虑，按照概率决定一个分类。

(5)待分类样本在到达属性a节点时就终止分类，然后根据此时a节点所覆盖的叶子节点类别状况为其分配一个发生概率最高的类。

推理规则

C4.5决策树能够根据决策树生成一系列规则集,我们可以把一颗决策树看成一系列规则的组合。一个规则对应着从根节点到叶子节点的路径，该规则的条件是路径上的条件，结果是叶子节点的类别。C4.5首先根据决策树的每个叶子节点生成一个规则集，对于规则集中的每条规则，算法利用“爬山”搜索来尝试是否有条件可以移除，由于移除一个条件和剪枝一个内部节点本质上是一样的，因此前面提到的悲观剪枝算法也被用在这里进行规则简化。MDL准则在这里也可以用来衡量对规则进行编码的信息量和对潜在的规则进行排序。简化后的规则数目要远远小于决策树的叶子节点数。根据简化后的规则集是无法重构原来的决策树的。规则集相比决策树而言更具有可操作性，因此在很多情况下我们需要从决策树中推理出规则集。C4.5有个缺点就是如果数据集增大了一点，那么学习时间会有一个迅速地增长。

数据挖掘十大算法--K-均值聚类算法

> 来源：http://blog.csdn.net/u011067360/article/details/24383051

一、相异度计算

在正式讨论聚类前，我们要先弄清楚一个问题：如何定量计算两个可比较元素间的相异度。用通俗的话说，相异度就是两个东西差别有多大，例如人类与章鱼的相异度明显大于人类与黑猩猩的相异度，这是能我们直观感受到的。但是，计算机没有这种直观感受能力，我们必须对相异度在数学上进行定量定义。

设 $X=\{x_1,x_2,...,x_n\},Y=\{y_1,y_2,...,y_n\}$，其中X，Y是两个元素项，各自具有n个可度量特征属性，那么X和Y的相异度定义为：

$d(X,Y)=f(X,Y)\rightarrow R$，其中R为实数域。也就是说相异度是两个元素对实数域的一个映射，所映射的实数定量表示两个元素的相异度。

下面介绍不同类型变量相异度计算方法。

1、标量

（1）标量也就是无方向意义的数字，也叫标度变量。现在先考虑元素的所有特征属性都是标量的情况。例如，计算X={2,1,102}和Y={1,3,2}的相异度。一种很自然的想法是用两者的欧几里得距离来作为相异度，欧几里得距离的定义如下：

$$d(X,Y)=\sqrt{(x_1-y_1)^2+(x_2-y_2)^2+...+(x_n-y_n)^2}$$

其意义就是两个元素在欧氏空间中的集合距离，因为其直观易懂且可解释性强，被广泛用于标识两个标量元素的相异度。将上面两个示例数据代入公式，可得两者的欧氏距离为：

$$d(X,Y)=\sqrt{(2-1)^2+(1-3)^2+(102-2)^2}=100.025$$

除欧氏距离外，常用作度量标量相异度的还有曼哈顿距离和闵可夫斯基距离，两者定义如下：

（2）曼哈顿距离：

$$d(X,Y)=|x_1-y_1|+|x_2-y_2|+...+|x_n-y_n|$$

（3） 闵可夫斯基距离：

$$d(X,Y)=\sqrt[p]{|x_1-y_1|^p+|x_2-y_2|^p+...+|x_n-y_n|^p}$$

（4）皮尔逊系数(Pearson Correlation Coefficient)

两个变量之间的皮尔逊相关系数定义为两个变量之间的协方差和标准差的商.

$$\rho_{X,Y} = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$$

(其中，E为数学期望或均值，D为方差，D开根号为标准差，E{ [X-ux] [Y-uy]}称为随机变量X与Y的协方差，记为Cov(X,Y)，即Cov(X,Y) = E{ [X-ux] [Y-ux]}，而两个变量之间的协方差和标准差的商则称为随机变量X与Y的相关系数，记为ρ_{XY}

欧氏距离和曼哈顿距离可以看做是闵可夫斯基距离在p=2和p=1下的特例。另外这三种距离都可以加权，这个很容易理解。

下面要说一下标量的规格化问题。上面这样计算相异度的方式有一点问题，就是取值范围大的属性对距离的影响高于取值范围小的属性。例如上述例子中第三个属性的取值跨度远大于前两个，这样不利于真实反映真实的相异度，为了解决这个问题，一般要对属性值进行规格化。

所谓规格化就是将各个属性值按比例映射到相同的取值区间，这样是为了平衡各个属性对距离的影响。通常将各个属性均映射到[0,1]区间，映射公式为：

$$a_i' = \frac{a_i - min(a_i)}{max(a_i) - min(a_i)}$$

其中max(ai)和min(ai)表示所有元素项中第i个属性的最大值和最小值。例如，将示例中的元素规格化到[0,1]区间后，就变成了X’={1,0,1}，Y’={0,1,0}，重新计算欧氏距离约为1.732。

2、二元变量

所谓二元变量是只能取0和1两种值变量，有点类似布尔值，通常用来标识是或不是这种二值属性。对于二元变量，上一节提到的距离不能很好标识其相异度，我们需要一种更适合的标识。一种常用的方法是用元素相同序位同值属性的比例来标识其相异度。

设有X={1,0,0,0,1,0,1,1}，Y={0,0,0,1,1,1,1,1}，可以看到，两个元素第2、3、5、7和8个属性取值相同，而第1、4和6个取值不同，那么相异度可以标识为3/8=0.375。一般的，对于二元变量，相异度可用“取值不同的同位属性数/单个元素的属性位数”标识。

上面所说的相异度应该叫做对称二元相异度。现实中还有一种情况，就是我们只关心两者都取1的情况，而认为两者都取0的属性并不意味着两者更相似。例如在根据病情对病人聚类时，如果两个人都患有肺癌，我们认为两个人增强了相似度，但如果两个人都没患肺癌，并不觉得这加强了两人的相似性，在这种情况下，改用“取值不同的同位属性数/(单个元素的属性位数-同取0的位数)”来标识相异度，这叫做非对称二元相异度。如果用1减去非对称二元相异度，则得到非对称二元相似度，也叫Jaccard系数，是一个非常重要的概念。

3、分类变量

分类变量是二元变量的推广，类似于程序中的枚举变量，但各个值没有数字或序数意义，如颜色、民族等等，对于分类变量，用“取值不同的同位属性数/单个元素的全部属性数”来标识其相异度。

4、序数变量

序数变量是具有序数意义的分类变量，通常可以按照一定顺序意义排列，如冠军、亚军和季军。对于序数变量，一般为每个值分配一个数，叫做这个值的秩，然后以秩代替原值当做标量属性计算相异度。

5、向量

对于向量，由于它不仅有大小而且有方向，所以闵可夫斯基距离不是度量其相异度的好办法，一种流行的做法是用两个向量的余弦度量，其度量公式为：

$$s(X,Y) = \frac{X^tY}{||X||||Y||}$$

其中||X||表示X的欧几里得范数。要注意，余弦度量度量的不是两者的相异度，而是相似度！

二、聚类问题

所谓聚类问题，就是给定一个元素集合D，其中每个元素具有n个可观察属性，使用某种算法将D划分成k个子集，要求每个子集内部的元素之间相异度尽可能低，而不同子集的元素相异度尽可能高。其中每个子集叫做一个簇。 与分类不同，分类是示例式学习，要求分类前明确各个类别，并断言每个元素映射到一个类别，而聚类是观察式学习，在聚类前可以不知道类别甚至不给定类别数量，是无监督学习的一种。目前聚类广泛应用于统计学、生物学、数据库技术和市场营销等领域，相应的算法也非常的多。本文仅介绍一种最简单的聚类算法——k均值（k-means）算法。

1、算法简介

k-means算法，也被称为k-平均或k-均值，是一种得到最广泛使用的聚类算法。 它是将各个聚类子集内的所有数据样本的均值作为该聚类的代表点，

算法的主要思想是通过迭代过程把数据集划分为不同的类别，使得评价聚类性能的准则函数达到最优，从而使生成的每个聚类内紧凑，类间独立。这一算法不适合处理离散型属性，但是对于连续型具有较好的聚类效果。

2、算法描述

1、为中心向量c1, c2, …, ck初始化k个种子

2、分组:

（1）将样本分配给距离其最近的中心向量

（2）由这些样本构造不相交（ non-overlapping ）的聚类

3、确定中心:

用各个聚类的中心向量作为新的中心

4、重复分组和确定中心的步骤，直至算法收敛。

3、算法 k-means算法

输入：簇的数目k和包含n个对象的数据库。

输出：k个簇，使平方误差准则最小。

算法步骤：

1.为每个聚类确定一个初始聚类中心，这样就有K 个初始聚类中心。

2.将样本集中的样本按照最小距离原则分配到最邻近聚类

3.使用每个聚类中的样本均值作为新的聚类中心。

4.重复步骤2.3直到聚类中心不再变化。

5.结束，得到K个聚类

PS

1、将样本分配给距离它们最近的中心向量，并使目标函数值减小

$$\sum_{i=1}^{n} \min_{j \in \{1,2,\ldots,k\}} \left|\left| \mathbf{x}_i - \mathbf{p}_j \right|\right|^2$$

2、更新簇平均值

$$\overline{X_i} = \frac{1}{|C_i|} \sum_{x \in C_i} X$$

3、计算准则函数E

$$E = \sum_{i=1}^{k} \sum_{x \in C_i} \left| X - \overline{X_i} \right|^2$$

4、划分聚类方法对数据集进行聚类时包括如下

三个要点：

（1）选定某种距离作为数据样本间的相似性度量

上面讲到，k-means聚类算法不适合处理离散型属性，对连续型属性比较适合。因此在计算数据样本之间的距离时，可以根据实际需要选择欧式距离、曼哈顿距离或者明考斯距离中的一种来作为算法的相似性度量，其中最常用的是欧式距离。下面我再给大家具体介绍一下欧式距离。平均值

假设给定的数据集 $X = \{x_m | m = 1,2,\ldots,total\}$ ，X中的样本用d个描述属性A1,A2…Ad来表示，并且d个描述属性都是连续型属性。数据样本xi=(xi1,xi2,…xid),xj=(xj1,xj2,…xjd)其中，xi1,xi2,…xid和xj1,xj2,…xjd分别是样本xi和xj对应d个描述属性A1,A2,…Ad的具体取值。样本xi和xj之间的相似度通常用它们之间的距离d(xi,xj)来表示，距离越小，样本xi和xj越相似，差异度越小；距离越大，样本xi和xj越不相似，差异度越大。

欧式距离公式如下：

$$d(x_i, x_j) = \sqrt{\sum_{k=1}^{d} (x_{ik} - x_{jk})^2}$$

（2）选择评价聚类性能的准则函数

k-means聚类算法使用误差平方和准则函数来 评价聚类性能。给定数据集X，其中只包含描述属性，不包含类别属性。假设X包含k个聚类子集X1,X2,…XK；各个聚类子集中的样本数量分别为n1，n2,…,nk;各个聚类子集的均值代表点（也称聚类中心）分别为m1，m2,…,mk。则误差平方和准则函数公式为：

$$E = \sum_{i=1}^{k} \sum_{p \in X_i} \left\| p - m_i \right\|^2$$

（3）相似度的计算根据一个簇中对象的平均值来进行。

1）将所有对象随机分配到k个非空的簇中。

2）计算每个簇的平均值，并用该平均值代表相应的簇。

3）根据每个对象与各个簇中心的距离，分配给最近的簇。

4）然后转2），重新计算每个簇的平均值。这个过程不断重复直到满足某个准则函数才停止

三、聚类例子

O	x	y
1	0	2
2	0	0
3	1.5	0
4	5	0
5	5	2

数据对象集合S见上表，作为一个聚类分析的二维样本，要求的簇的数量k=2。

(1)选择 $O_1(0,2)$ ， $O_2(0,0)$ 为初始的簇中心，即 $M_1 = O_1 = (0,2)$ ， $M_2 = O_2 = (0,0)$ 。

(2)对剩余的每个对象，根据其与各个簇中心的距离，将它赋给最近的簇

对O3 ：

$$d(M_1, O_3) = \sqrt{(0-1.5)^2 + (2-0)^2} = 2.5$$

$$d(M_2, O_3) = \sqrt{(0-1.5)^2 + (0-0)^2} = 1.5$$

显然 $d(M_2, O_3) \le d(M_1, O_3)$ O3，故将C2分配给

对于O4:

$$d(M_1, O_4) = \sqrt{(0-5)^2 + (2-0)^2} = \sqrt{29}$$

$$d(M_2, O_4) = \sqrt{(0-5)^2 + (0-0)^2} = 5$$

因为： $d(M_2, O_4) \le d(M_1, O_4)$ 所以将O4分配给C2

对于O5:

$$d(M_1, O_5) = \sqrt{(0-5)^2 + (2-2)^2} = 5$$

$$d(M_2, O_5) = \sqrt{(0-5)^2 + (0-2)^2} = \sqrt{29}$$

因为： $d(M_1, O_5) \le d(M_2, O_5)$ 所以讲O5分配给C1

更新，得到新簇 $C_1 = \{O_1, O_5\}$；和 $C_2 = \{O_2, O_3, O_4\}$

计算平方误差准则，单个方差为

$$E_1 = \left[(0-0)^2 + (2-2)^2\right] + \left[(0-5)^2 + (2-2)^2\right] = 25 \quad M_1 = O_1 = (0,2)$$

$$E_2 = 27.25 \quad M_2 = O_2 = (0,0)$$

总体平均方差是：

$$E = E_1 + E_2 = 25 + 27.25 = 52.25$$

（3）计算新的簇的中心。

$$M_1 = ((0+5)/2, (2+2)/2) = (2.5, 2)$$
$$M_2 = ((0+1.5+5)/3, (0+0+0)/3) = (2.17, 0)$$

重复（2）和（3），得到O1分配给C1；O2分配给C2，O3分配给C2，O4分配给C2，O5分配给C1。更新，得到新簇 $C_1 = \{O_1, O_5\}$ 和 $C_2 = \{O_2, O_3, O_4\}$ 。中心为 $M_1 = (2.5, 2)$ ，$M_2 = (2.17, 0)$ 。

单个方差分别为

$$E_1 = \left[(0-2.5)^2 + (2-2)^2\right] + \left[(2.5-5)^2 + (2-2)^2\right] = 12.5 \quad E_2 = 13.15$$

总体平均误差是：

$$E = E_1 + E_2 = 12.5 + 13.15 = 25.65$$

由上可以看出，第一次迭代后，总体平均误差值52.25~25.65，显著减小。由于在两次迭代中，簇中心不变，所以停止迭代过程，算法停止。

PS

1、k-means算法的性能分析

主要优点：

是解决聚类问题的一种经典算法，简单、快速。

对处理大数据集，该算法是相对可伸缩和高效率的。因为它的复杂度是0 (n k t) , 其中, n 是所有对象的数目, k 是簇的数目, t 是迭代的次数。通常k < <n 且t < <n 。

当结果簇是密集的，而簇与簇之间区别明显时, 它的效果较好。

主要缺点

在簇的平均值被定义的情况下才能使用，这对于处理符号属性的数据不适用。

必须事先给出k（要生成的簇的数目），而且对初值敏感，对于不同的初始值，可能会导致不同结果。

它对于“躁声”和孤立点数据是敏感的，少量的该类数据能够对平均值产生极大的影响。

K-Means算法对于不同的初始值，可能会导致不同结果。解决方法：

1.多设置一些不同的初值，对比最后的运算结果）一直到结果趋于稳定结束，比较耗时和浪费资源

2.很多时候，事先并不知道给定的数据集应该分成多少个类别才最合适。这也是 K-means 算法的一个不足。有的算法是通过类的自动合并和分裂，得到较为合理的类型数目 K.

2、k-means算法的改进方法——k-prototype算法

k-Prototype算法：可以对离散与数值属性两种混合的数据进行聚类，在k-prototype中定义了一个对数值与离散属性都计算的相异性度量标准。

K-Prototype算法是结合K-Means与K-modes算法，针对混合属性的，解决2个核心问题如下：

1.度量具有混合属性的方法是，数值属性采用K-means方法得到P1，分类属性采用K-modes方法P2，那么D=P1+a*P2，a是权重，如果觉得分类属性重要，则增加a，否则减少a，a=0时即只有数值属性

2.更新一个簇的中心的方法，方法是结合K-Means与K-modes的更新方法。

3、k-means算法的改进方法——k-中心点算法

k-中心点算法：k -means算法对于孤立点是敏感的。为了解决这个问题，不采用簇中的平均值作为参照点，可以选用簇中位置最中心的对象，即中心点作为参照点。这样划分方法仍然是基于最小化所有对象与其参照点之间的相异度之和的原则来执行的。

机器学习与数据挖掘-支持向量机(SVM)

机器学习与数据挖掘-支持向量机(SVM)（一）

> 来源：http://blog.csdn.net/u011067360/article/details/24876849

最近在看斯坦福大学的机器学习的公开课，学习了支持向量机，再结合网上各位大神的学习经验总结了自己的一些关于支持向量机知识。

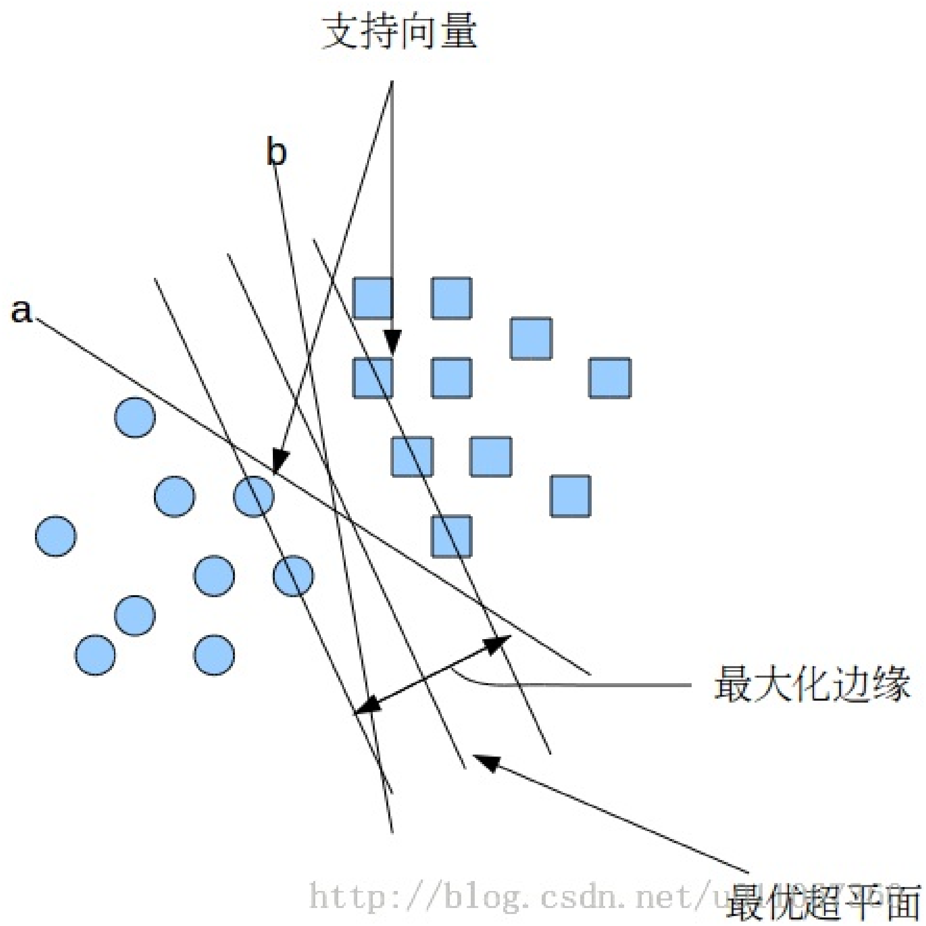
一、什么是支持向量机(SVM)?

1、支持向量机（Support Vector Machine，常简称为SVM）是一种监督式学习的方法，可广泛地应用于统计分类以及回归分析。支持向量机属于一般化线性分类器，这族分类器的特点是他们能够同时最小化经验误差与最大化几何边缘区，因此支持向量机也被称为最大边缘区分类器。

2、支持向量机将向量映射到一个更高维的空间里，在这个空间里建立有一个最大间隔超平面。在分开数据的超平面的两边建有两个互相平行的超平面，分隔超平面使两个平行超平面的距离最大化。假定平行超平面间的距离或差距越大，分类器的总误差越小。

3、假设给定一些分属于两类的2维点，这些点可以通过直线分割， 我们要找到一条最优的分割线，如何来界定一个超平面是不是最优的呢?

如下图：



在上面的图中，a和b都可以作为分类超平面，但最优超平面只有一个，最优分类平面使间隔最大化。 那是不是某条直线比其他的更加合适呢? 我们可以凭直觉来定义一条评价直线好坏的标准:

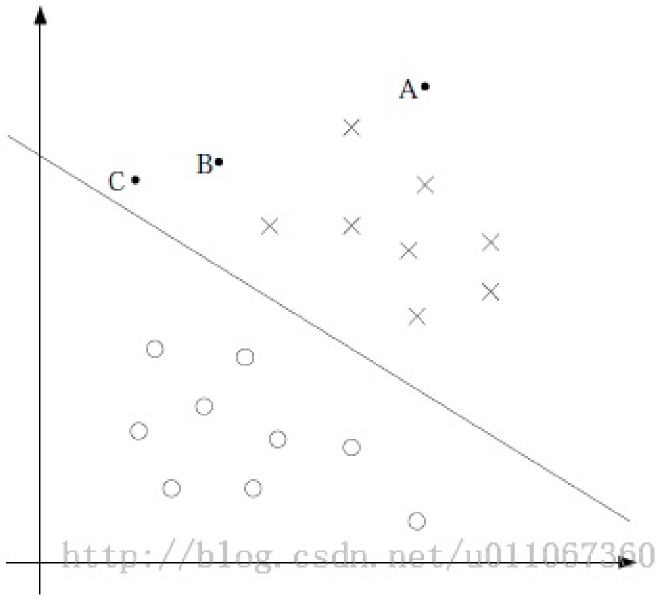
距离样本太近的直线不是最优的，因为这样的直线对噪声敏感度高，泛化性较差。 因此我们的目标是找到一条直线（图中的最优超平面），离所有点的距离最远。 由此， SVM算法的实质是找出一个能够将某个值最大化的超平面，这个值就是超平面离所有训练样本的最小距离。这个最小距离用SVM术语来说叫做间隔(margin) 。

二、如何计算最优超平面?

1、线性分类：

我们通常希望分类的过程是一个机器学习的过程。这些数据点并不需要是$\mathbb{R}^2$中的点，而可以是任意$\mathbb{R}^n$的点（一个超平面，在二维空间中的例子就是一条直线）。我们希望能够把这些点通过一个n-1维的超平面分开，通常这个被称为线性分类器。有很多分类器都符合这个要求，但是我们还希望找到分类最佳的平面，即使得属于两个不同类的数据点间隔最大的那个面，该面亦称为最大间隔超平面。如果我们能够找到这个面，那么这个分类器就称为最大间隔分类器。

我们从下面一个图开始：

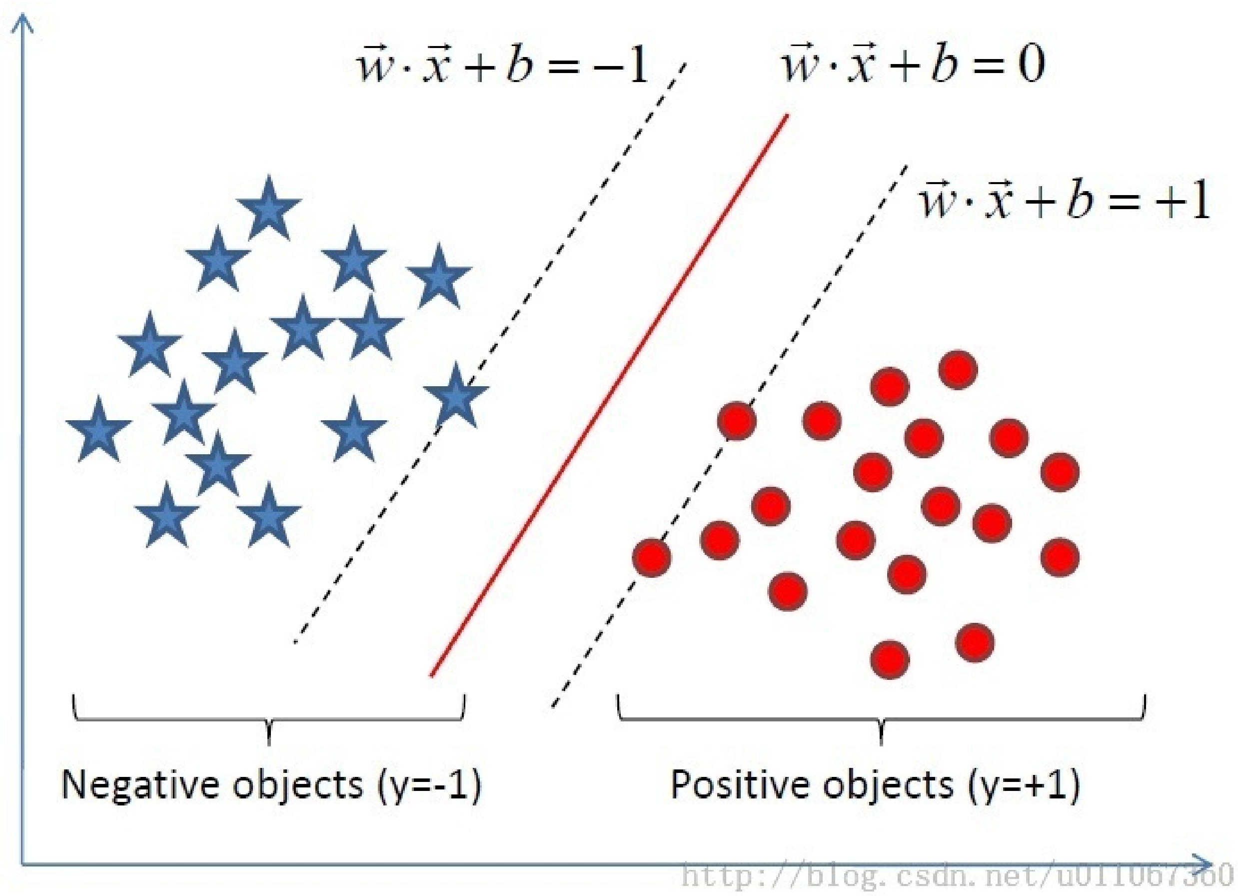


中间那条线是wx + b =0，我们强调所有点尽可能地远离中间那条线。考虑上面3个点A、B和C。从图中我们可以确定A是×类别的，然而C我们是不太确定的，B还算能够确定。这样我们可以得出结论，我们更应该关心靠近中间分割线的点，让他们尽可能地远离中间线，而不是在所有点上达到最优。因为那样的话，要使得一部分点靠近中间线来换取另外一部分点更加远离中间线。同时这个所谓的超平面的的确把这两种不同形状的数据点分隔开来，在超平面一边的数据点所对应的 y 全是 -1 ，而在另一边全是 1 。

我们可以令分类函数：

$$f(x) = w^T x + b$$

显然，如果 f(x)=0 ，那么 x 是位于超平面上的点。我们不妨要求对于所有满足 f(x)<0 的点，其对应的="" y="" 等于="" -1="" ，而="" f(x)="">0 则对应 y=1 的数据点。如下图。



最优超平面可以有无数种表达方式，即通过任意的缩放 **w** 和 **b** 。 习惯上我们使用以下方式来表达最优超平面

$w^T x + b$ **=1**

式中 x 表示离超平面最近的那些点，也可以就可以得到支持向量的表达式为：y(wx + b) = 1,

上面说了，我们令两类的点分别为+1, -1，所以当有一个新的点x需要预测属于哪个分类的时候，我们用sgn(f(x))，就可以预测了，sgn表示符号函数，当f(x) > 0的时候，sgn(f(x)) = +1, 当f(x) < 0的时候sgn(f(x)) = –1。

通过几何学的知识，我们知道点 x 到超平面 (β, β_0) 的距离为:

$$\gamma = \frac{w^T x + b}{||w||} = \frac{f(x)}{||w||}$$

特别的， 对于超平面, 表达式中的分子为1，因此支持向量到超平面的距离是

$$\text{distance}_{\text{support vectors}}$$

||w||的意思是w的二范数。

刚才我们介绍了间隔(margin),这里表示为 M, 它的取值是最近距离的2倍:

```
M = 2 / ||w||
```

最大化这个式子等价于最小化||w||, 另外由于||w||是一个单调函数，我们可以对其加入平方，和前面的系数，熟悉的同学应该很容易就看出来了，这个式子是为了方便求导。

最后最大化 M 转化为在附加限制条件下最小化函数：

$$\max \frac{1}{\|w\|} \quad \to \quad \min \frac{1}{2}\|w\|^2$$

即：$$\max \frac{1}{\|w\|}, \quad s.t., y_i(w^T x_i + b) \geq 1, i = 1, \ldots, n$$

这是一个拉格朗日优化问题，可以通过拉格朗日乘数法得到最优超平面的权重向量**W**和偏置 **b** 。

$$L(w,b,a) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i (y_i (w^T x_i + b) - 1)$$

PS

1、咱们就要确定上述分类函数f(x) = w.x + b（w.x表示w与x的内积）中的两个参数w和b，通俗理解的话w是法向量，b是截距;

2、那如何确定w和b呢？答案是寻找两条边界端或极端划分直线中间的最大间隔（之所以要寻最大间隔是为了能更好的划分不同类的点，下文你将看到：为寻最大间隔，导出1/2||w||^2，继而引入拉格朗日函数和对偶变量a，化为对单一因数对偶变量a的求解，当然，这是后话），从而确定最终的最大间隔分类超平面hyper plane和分类函数；

3、进而把寻求分类函数f(x) = w.x + b的问题转化为对w，b的最优化问题，最终化为对偶因子的求解。

支持向量机(SVM)（二）-- 拉格朗日对偶（Lagrange duality）

> 来源：http://blog.csdn.net/u011067360/article/details/25215465

简介：

1、在之前我们把要寻找最优的分割超平面的问题转化为带有一系列不等式约束的优化问题。这个最优化问题被称作原问题。我们不会直接解它，而是把它转化为对偶问题进行解决。

2、为了使问题变得易于处理，我们的方法是把目标函数和约束全部融入一个新的函数，为了使问题变得易于处理，我们的方法是把目标函数和约束全部融入一个新的函数，即拉格朗日函数，再通过这个函数来寻找最优点。即拉格朗日函数，再通过这个函数来寻找最优点。

3、约束条件可以分成不等式约束条件和等式约束条件，只有等式约束条件的问题我们在高等数学课程中已经学习过了，其解决方法是直接将等式约束加入原问题构造出拉格朗日函数，然后求导即可。现在考虑更加一般性的问题：带不等式约束和等式约束的极值问题如何构造拉格朗日函数求解。 学习拉格朗日对偶原理重要的是理解构造所得的原始问题和原函数的等价性，以及原始问题和对偶问题解得等价性。

回忆一下之前得到的目标函数

$$\min \frac{1}{2}\|w\|^2 \quad s.t., y_i(w^Tx_i+b)\geq 1, i=1,\ldots,n$$

要求解这个式子。我们可以通过求解对偶问题得到最优解，这就是线性可分条件下支持向量机的对偶算法，这样做的优点在于：一者对偶问题往往更容易求解；二者可以自然的引入核函数，进而推广到非线性分类问题。

1、我们下看下面的一个式子

$$\begin{aligned}\min_w \quad & f(w)\\ \text{s.t.} \quad & h_i(w)=0,\quad i=1,\ldots,l.\end{aligned}$$

上面是要求解的目标函数及其条件。我们可以得到拉格朗日公式为

$$\mathcal{L}(w,\beta)=f(w)+\sum_{i=1}^{l}\beta_i h_i(w)$$

在这里 w 被称为拉格朗日算子。

然后分别对w和求偏导，使得偏导数等于0，然后解出 w 和 β；

$$\frac{\partial \mathcal{L}}{\partial w_i} = 0; \quad \frac{\partial \mathcal{L}}{\partial \beta_i} = 0.$$

下面我们将要产生一个既有等式又有不等式条件限制的式子，我们可以叫做原始优化问题，这里简单介绍下拉格朗日对偶的原理。如下式子：

$$\begin{aligned} \min_w \quad & f(w) \\ \text{s.t.} \quad & g_i(w) \le 0, \quad i = 1, \ldots, k \\ & h_i(w) = 0, \quad i = 1, \ldots, l. \end{aligned}$$

上述式子有两个条件（等式条件和不等式条件）

由此我们定义一般化的拉格朗日公式

$$\mathcal{L}(w, \alpha, \beta) = f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{i=1}^{l} \beta_i h_i(w).$$

这里的α_i和β_i都是拉格朗日算子。不要忘了我们求解的是最小值。

设如下等式：

$$\theta_{\mathcal{P}}(w) = \max_{\alpha, \beta : \alpha_i \ge 0} \mathcal{L}(w, \alpha, \beta).$$

这里的P代表primal。我们设如下约束条件（primal constraints）：

$$\begin{aligned} g_i(w) &\le 0, \quad i = 1, \ldots, k \\ h_i(w) &= 0, \quad i = 1, \ldots, l. \end{aligned}$$

如果条件不全部满足的话，我们总可以调整α_i和β_i使最大值出现正无穷，即会出现下面情况：

$$\begin{aligned} \theta_{\mathcal{P}}(w) &= \max_{\alpha, \beta : \alpha_i \ge 0} f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{i=1}^{l} \beta_i h_i(w) \\ &= \infty. \end{aligned}$$

因此我们可以得出如下式子：

$$\theta_{\mathcal{P}}(w) = \begin{cases} f(w) & \text{if } w \text{ satisfies primal constraints} \\ \infty & \text{otherwise.} \end{cases}$$

这样我们原来要求的min f(w)可以转换成求了。

$$\min_w \theta_{\mathcal{P}}(w) = \min_w \max_{\alpha, \beta : \alpha_i \ge 0} \mathcal{L}(w, \alpha, \beta),$$ 同时我们设

$$\theta_{\mathcal{D}}(\alpha,\beta)=\min_{w}\mathcal{L}(w,\alpha,\beta).$$

$\theta_{\mathcal{D}}(\alpha,\beta)$将问题转化为先求拉格朗日关于$w$的最小值，这个时候就把$\alpha$和$\beta$看做常量。之后再求最大值。

如下：

$$\max_{\alpha,\beta:\alpha_i\geq 0}\theta_{\mathcal{D}}(\alpha,\beta)=\max_{\alpha,\beta:\alpha_i\geq 0}\min_{w}\mathcal{L}(w,\alpha,\beta).$$

这个问题就是原问题的对偶问题，相对于原问题只是更换了min和max的顺序，而一般更换顺序的结果是Max Min(X) <= Min Max(X)。然而在这里两者相等。由此我们可以设如下：

$$d^*=\max_{\alpha,\beta:\alpha_i\geq 0}\min_{w}\mathcal{L}(w,\alpha,\beta)\leq\min_{w}\max_{\alpha,\beta:\alpha_i\geq 0}\mathcal{L}(w,\alpha,\beta)=p^*.$$

所以在一定的条件下我们可以得到：

$$d^*=p^*.$$

因此我们可以解决原问题的对偶问题，但是我们还要看看条件是什么。

假设f和g都是凸函数，h是仿射的(仿射的含义：存在ai, bi，使得$h_i(w)=a_i^Tw+b_i$.，并且还要满足存在w使得所有的i都有$g_i(w)<0$；

有了以上的假设，那么一定会存在使得w^*是原问题的解，α^*,β^*是对偶问题的解。同时也满足

$p^*=d^*=\mathcal{L}(w^*,\alpha^*,\beta^*)$.另外，$w^*,\alpha^*,\beta^*$还满足Karush-Kuhn-Tucker（KKT condition），如下式子：

$$\begin{aligned}
\frac{\partial}{\partial w_i}\mathcal{L}(w^*,\alpha^*,\beta^*) &= 0,\quad i=1,\ldots,n\\
\frac{\partial}{\partial \beta_i}\mathcal{L}(w^*,\alpha^*,\beta^*) &= 0,\quad i=1,\ldots,l\\
\alpha_i^*g_i(w^*) &= 0,\quad i=1,\ldots,k\\
g_i(w^*) &\leq 0,\quad i=1,\ldots,k\\
\alpha^* &\geq 0,\quad i=1,\ldots,k
\end{aligned}$$

所以w^*,α^*,β^*如果满足了KKT条件，那么他们就是原问题和对偶问题的解。

我们从式子 $\alpha_i^*g_i(w^*)=0,\ i=1,\ldots,k$ 和式子 $g_i(w^*)\leq 0,\ i=1,\ldots,k$ 可以看出如果$\alpha_i^*>0$,那么 $g_i(w^*)=0$,

这个也就说明 $g_i(w^*) = 0$时，w处于可行域的边界上，这时才是起作用的约束。

而其他位于可行域内部的（ $g_i(w^*) < 0$ ）点都是不起作用的约束，其中也就是$\alpha^* = 0$的时候。这个KKT双重补足条件会用来解释支持向量和SMO的收敛测试。

支持向量机(SVM)（三）-- 最优间隔分类器（optimal margin classifier）

> 来源：http://blog.csdn.net/u011067360/article/details/25322637

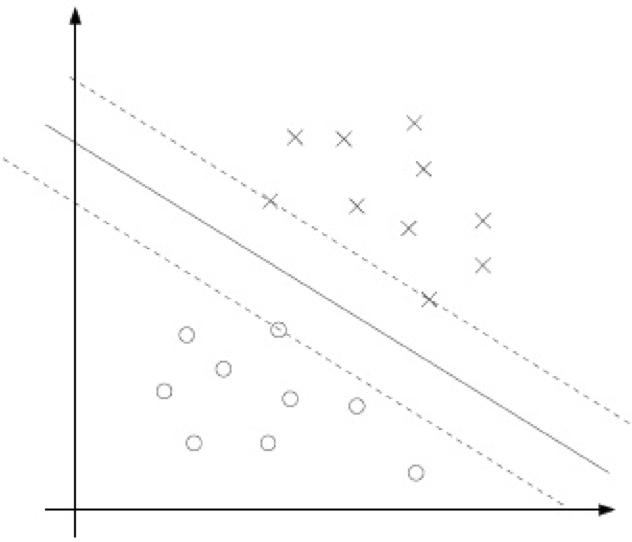
在之前为了寻找最有分类器，我们提出了如下优化问题：

$$\begin{aligned} \min_{\gamma,w,b} \quad & \frac{1}{2}||w||^2 \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1, \quad i = 1, \ldots, m \end{aligned}$$

在这里我们可以把约束条件改写成如下：

$$g_i(w) = -y^{(i)}(w^T x^{(i)} + b) + 1 \leq 0.$$

首先我们看下面的图示：



很显然我们可以看出实线是最大间隔超平面，假设×号的是正例，圆圈的是负例。在虚线上的点和在实线上面的两个一共这三个点称作支持向量。现在我们结合KKT条件分析下这个图。

$$\begin{aligned} \frac{\partial}{\partial w_i}\mathcal{L}(w^*, \alpha^*, \beta^*) &= 0, \quad i = 1, \ldots, n \\ \frac{\partial}{\partial \beta_i}\mathcal{L}(w^*, \alpha^*, \beta^*) &= 0, \quad i = 1, \ldots, l \\ \alpha_i^* g_i(w^*) &= 0, \quad i = 1, \ldots, k \\ g_i(w^*) &\leq 0, \quad i = 1, \ldots, k \\ \alpha^* &\geq 0, \quad i = 1, \ldots, k \end{aligned}$$

我们从式子 $\alpha_i^* g_i(w^*) = 0, \; i = 1, \ldots, k$ 和式子 $g_i(w^*) \le 0, \; i = 1, \ldots, k$ 可以看出如果 $\alpha_i^* > 0$,那么 $g_i(w^*) = 0$,

这个也就说明 $g_i(w^*) = 0$ 时，w处于可行域的边界上，这时才是起作用的约束。

1、那我们现在可以构造拉格朗日函数如下：

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{m} \alpha_i \left[y^{(i)} (w^T x^{(i)} + b) - 1 \right].$$

注意到这里只有 α_i 没 β_i 是因为原问题中没有等式约束，只有不等式约束。

2、接下来我们对w和b分别求偏导数。

$$\nabla_w \mathcal{L}(w, b, \alpha) = w - \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} = 0$$

$$\frac{\partial}{\partial b} \mathcal{L}(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i y^{(i)} = 0.$$

并得到

$$w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}.$$

3、将上式带回到拉格朗日函数中得到：

$$\mathcal{L}(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j (x^{(i)})^T x^{(j)} - b \sum_{i=1}^{m} \alpha_i y^{(i)}.$$

由于 $\frac{\partial}{\partial b} \mathcal{L}(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i y^{(i)} = 0.$ ，因此简化为

$$\mathcal{L}(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j (x^{(i)})^T x^{(j)}.$$

4、现在我们得到了关于w和b的可以最小化的等式，我们在联合 α_i 这个参数，当然他的条件还 α_i>=0，现在我们可以得到如下的二元优化等式了：

$$\begin{aligned} \max_{\alpha} \quad & W(\alpha)=\sum_{i=1}^{m} \alpha_i-\frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle. \\ \text{s.t.} \quad & \alpha_i \geq 0, \quad i=1, \ldots, m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)}=0, \end{aligned}$$

5、现在你还必须知道我们之前讲解的条件一是 $d^*=p^*$，二是KKT条件：

$$d^*=\max_{\alpha,\beta:\alpha_i\geq 0} \min_{w} \mathcal{L}(w,\alpha,\beta) \leq \min_{w} \max_{\alpha,\beta:\alpha_i \geq 0} \mathcal{L}(w,\alpha,\beta)=p^*.$$

很显然存在w使得对于所有的i$g_i(w)<0$。因此，一定存w^*,α^*使w^*是原问题的解α^*是对偶问题的解。

如果求出α_i（也就是α^*），根据

$$w=\sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}.$$

即可求出w（也是w^*，原问题的解）。然后

$$b^*=-\frac{\max_{i:y^{(i)}=-1} w^{*T} x^{(i)}+\min_{i:y^{(i)}=1} w^{*T} x^{(i)}}{2}.$$

即可求出b。即离超平面最近的正的函数间隔要等于离超平面最近的负的函数间隔。

6、现在我们在看另外一个问题：

由

$$w=\sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}.$$

所

$$\begin{aligned} w^T x+b &= \left(\sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}\right)^T x+b \\ &= \sum_{i=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, x \rangle+b. \end{aligned}$$

这里我们将向量内$(x^{(i)})^T x^{(j)}$表示$\langle x^{(i)}, x^{(j)} \rangle$.

现在可以看出我要计算等式的话就只需要计算向量的内积就好了，同时要α_i在支持向量上面的话，那$\alpha_i=0$，这样就更简单了，因此很多的值都是0。

支持向量机（四）-- 核函数

> 来源：http://blog.csdn.net/u011067360/article/details/25503073

一、核函数的引入

问题1：

SVM显然是线性分类器，但数据如果根本就线性不可分怎么办？

解决方案1：

数据在原始空间（称为输入空间）线性不可分，但是映射到高维空间（称为特征空间）后很可能就线性可分了。

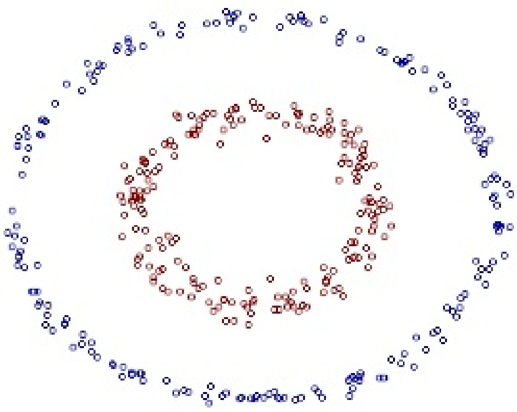
问题2：

映射到高维空间同时带来一个问题：在高维空间上求解一个带约束的优化问题显然比在低维空间上计算量要大得多，这就是所谓的“维数灾难”。

解决方案2：

于是就引入了“核函数”，核函数的价值在于它虽然也是讲特征进行从低维到高维的转换。

二、实例说明 例如图中的两类数据，分别分布为两个圆圈的形状，不论是任何高级的分类器，只要它是线性的，就没法处理，SVM 也不行。因为这样的数据本身就是线性不可分的。



从上图我们可以看出一个理想的分界应该是一个“圆圈”而不是一条线（超平面）。如果用 X1 和 X2 来表示这个二维平面的两个坐标的话，我们知道一条二次曲线（圆圈是二次曲线的一种特殊情况）的方程可以写作这样的形式：

```
a1X1+a2X21+a3X2+a4X22+a5X1X2+a6=0
```

注意上面的形式，如果我们构造另外一个五维的空间，其中五个坐标的值分别为 Z1=X1, Z2=X21, Z3=X2, Z4=X22, Z5=X1X2，那么显然，上面的方程在新的坐标系下可以写作：

```
∑i=15aiZi+a6=0
```

关于新的坐标 Z，这正是一个超平面 的方程！也就是说，如果我们做一个映射 ϕ:R2→R5，将 X 按照上面的规则映射为 Z，那么在新的空间中原来的数据将变成线性可分的，从而使用之前我们推导的线性分类算法就可以进行处理了。这正是 Kernel 方法处理非线性问题的基本思想。

三、详细分析

还记得之前我们用内积$\langle x^{(i)}, x^{(j)} \rangle$.这里是二维模型，但是现在我们需要三维或者更高的维度来表示样本。这里我们假设是维度是三；

那么首先需要将特征x扩展到三维(x, x^2, x^3)，然后寻找特征和结果之间的模型。我们将这种特征变换称作特征映射（feature mapping）。映射函数称作ϕ，在这个例子中

$$\phi(x) = \begin{bmatrix} x \\ x^2 \\ x^3 \end{bmatrix}$$

我们希望将得到的特征映射后的特征应用于SVM分类，而不是最初的特征。这样，我们需要将前面$w^T x + b$公式中的内积从$< x^{(i)}, x >$，映射到$< \phi(x^{(i)}), \phi(x) >$。

为什么需要映射后的特征而不是最初的特征来参与计算，一个重要原因是样例可能存在线性不可分的情况，而将特征映射到高维空间后，往往就可分了。

核函数的定义：

将核函数形式化定义，如果原始特征内积是$< x, z >$，映射后为$< \phi(x), \phi(z) >$，那么定义核函数（Kernel）为

$$K(x,z) = \phi(x)^T \phi(z)$$

现在有了以上的概念，我们现在要计算K(x,z)只要简单的计算$\phi(x)$，然后计算 $\phi(x)^T \phi(z)$，在求出它们的内积。但是现在有一个问题，那是计算K(x,z)的时间复杂度是提高了。即使是$\phi(x)$计算也是很复杂的。那现在怎么解决呢？

现在我们假设：x，z都是n维，同时有

$$K(x, z) = (x^T z)^2$$

展开

$$\begin{aligned} K(x,z) &= \left(\sum_{i=1}^{n} x_i z_i\right)\left(\sum_{j=1}^{n} x_j z_j\right) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j z_i z_j \\ &= \sum_{i,j=1}^{n} (x_i x_j)(z_i z_j) \end{aligned}$$

发现我们可以只计算原始特征x和z内积的平方（时间复杂度是O(n)），就等价与计算映射后特征的内积。也就是说我们不需要$O(n^2)$时间了。

现在看一下映射函数（n=3时），根据上面的公式，得到

$$\phi(x) = \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_1x_3 \\ x_2x_1 \\ x_2x_2 \\ x_2x_3 \\ x_3x_1 \\ x_3x_2 \\ x_3x_3 \end{bmatrix}.$$

也就是说核函数$K(x,z) = (x^T z)^2$只能在选择这样的ϕ作为映射函数时才能够等价于映射后特征的内积。

再看一个核函数

$$\begin{aligned} K(x,z) &= (x^T z + c)^2 \\ &= \sum_{i,j=1}^{n} (x_i x_j)(z_i z_j) + \sum_{i=1}^{n} (\sqrt{2c}x_i)(\sqrt{2c}z_i) + c^2. \end{aligned}$$

对应的映射函数（n=3时）是

$$\phi(x) = \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_1x_3 \\ x_2x_1 \\ x_2x_2 \\ x_2x_3 \\ x_3x_1 \\ x_3x_2 \\ x_3x_3 \\ \sqrt{2c}x_1 \\ \sqrt{2c}x_2 \\ \sqrt{2c}x_3 \\ c \end{bmatrix},$$

更一般地，核函数$K(x,z) = (x^Tz + c)^d$对应的映射后特征维度为$\binom{n+d}{d}$。

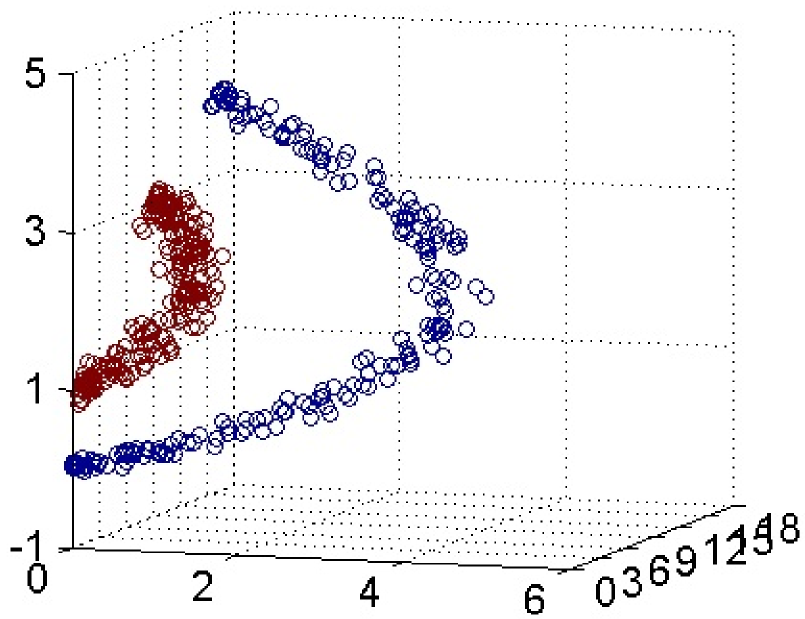
四、如何映射到核函数

现在介绍了核函数之后那到底怎么来使用核函数到样本了？

设超平面实际的方程是这个样子（圆心在 X2 轴上的一个正圆）：

```
a1X21+a2(X2-c)2+a3=0
```

因此我只需要把它映射到 Z1=X21, Z2=X22, Z3=X2 这样一个三维空间中即可，下图是映射之后的结果，将坐标轴经过适当的旋转，就可以很明显地看出，数据是可以通过一个平面来分开的：



现在让我们再回到 SVM 的情形，假设原始的数据时非线性的，我们通过一个映射 ϕ(·) 将其映射到一个高维空间中，数据变得线性可分了，这个时候，我们就可以使用原来的推导来进行计算，只是所有的推导现在是在新的空间，而不是原始空间中进行。

我们上一次得到的最终的分类函数是这样的：

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i \langle x_i, x \rangle + b$$

现在则是在映射过后的空间，即：

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i \langle \phi(x_i), \phi(x) \rangle + b$$

而其中的 α 也是通过求解如下 dual 问题而得到的：

$$\begin{aligned} &\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \phi(x_i), \phi(x_j) \rangle \\ &s.t., \alpha_i \geq 0, i = 1, \ldots, n \\ &\qquad \sum_{i=1}^{n} \alpha_i y_i = 0 \end{aligned}$$

回到我们之前构造的一个五维的空间：到现在貌似我们还没有用到核函数，但是现在我们可以看出，数据映射到新空间后，因为新空间是多维的，计算量肯定是增加了不少了，现在就只能用核函数来解决了。

$$\sum_{i=1}^{5} a_i Z_i + a_6 = 0$$

不妨还是从最开始的简单例子出发，设两个向量 $x_1 = (\eta_1, \eta_2)^T$ 和 $x_2 = (\xi_1, \xi_2)^T$ ，而 ϕ(*) 即是到前面说的五维空间的映射，

五个坐标的值分别为 Z1=X1, Z2=X21, Z3=X2, Z4=X22, Z5=X1X2,

因此映射过后的内积为：

$$\langle \phi(x_1), \phi(x_2) \rangle = \eta_1 \xi_1 + \eta_1^2 \xi_1^2 + \eta_2 \xi_2 + \eta_2^2 \xi_2^2 + \eta_1 \eta_2 \xi_1 \xi_2$$

根据我们之前简介的核函数的实现，具体来说，上面这个式子的计算结果实际上映射了

$$\varphi(X_1, X_2) = (\sqrt{2} X_1, X_1^2, \sqrt{2} X_2, X_2^2, \sqrt{2} X_1 X_2, 1)^T$$

这样一来计算的问题就算解决了，避开了直接在高维空间中进行计算，而结果却是等价的。

五、高斯核函数

再看另外一个核函数

$$K(x,z)=\exp\left(-\frac{||x-z||^2}{2\sigma^2}\right).$$

这时，如果x和z很相近（$||x-z||\approx 0$），那么核函数值为1，如果x和z相差很大（$||x-z||\gg 0$），那么核函数值约等于0。由于这个函数类似于高斯分布，因此称为高斯核函数，也叫做径向基函数(Radial Basis Function 简称RBF)。它能够把原始特征映射到无穷维。

既然高斯核函数能够比较x和z的相似度，并映射到0到1，回想logistic回归，sigmoid函数可以，因此还有sigmoid核函数等等。

注意，使用核函数后，怎么分类新来的样本呢？线性的时候我们使用SVM学习出w和b，新来样本x的话，我们使用$w^T x + b$来判断，如果值大于等于1，那么是正类，小于等于是负类。在两者之间，认为无法确定。如果使用了核函数后，$w^T x + b$就变成了$w^T\phi(x)+b$，是否先要找到$\phi(x)$，然后再预测？答案肯定不是了，找$\phi(x)$很麻烦，回想我们之前说过的

$$\begin{aligned} w^T x + b &= \left(\sum_{i=1}^m \alpha_i y^{(i)} x^{(i)}\right)^T x + b \\ &= \sum_{i=1}^m \alpha_i y^{(i)} \langle x^{(i)}, x\rangle + b. \end{aligned}$$

只需将$< x^{(i)}, x >$替换成$K(x^{(i)}, x)$，然后值的判断同上。

总结：对于非线性的情况，SVM 的处理方法是选择一个核函数 κ(*)，通过将数据映射到高维空间，来解决在原始空间中线性不可分的问题。由于核函数的优良品质，这样的非线性扩展在计算量上并没有比原来复杂多少，这一点是非常难得的。当然，这要归功于核方法——除了 SVM 之外，任何将计算表示为数据点的内积的方法，都可以使用核方法进行非线性扩展。

参考文档：（主要的参考文档来自4个地方）

1、支持向量机: Kernel

2、JerryLead关于核函数的讲解

3、 支持向量机通俗导论（理解SVM的三层境界

4、斯坦福大学机器学习的公开课。

支持向量机(SVM)（五）-- SMO算法详解

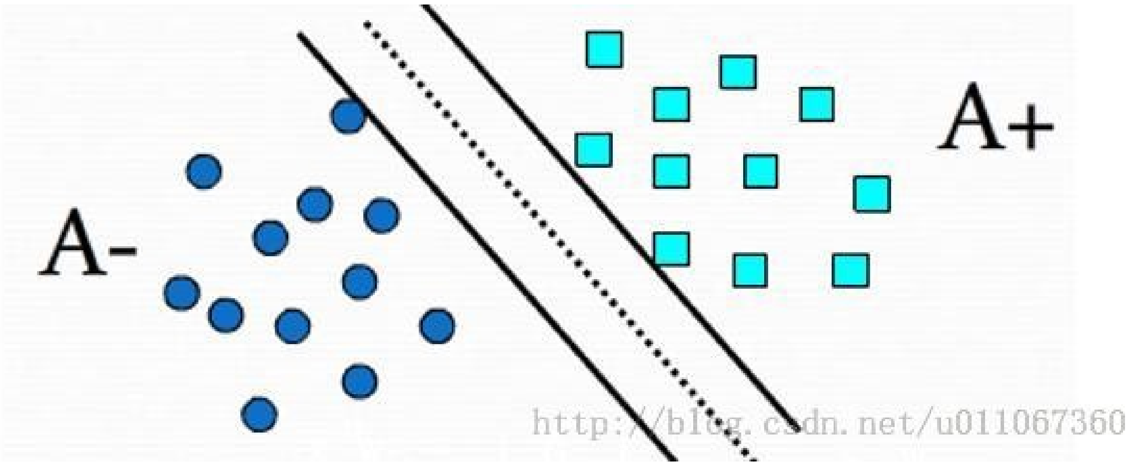
> 来源：http://blog.csdn.net/u011067360/article/details/26503719

一、我们先回顾下SVM问题。

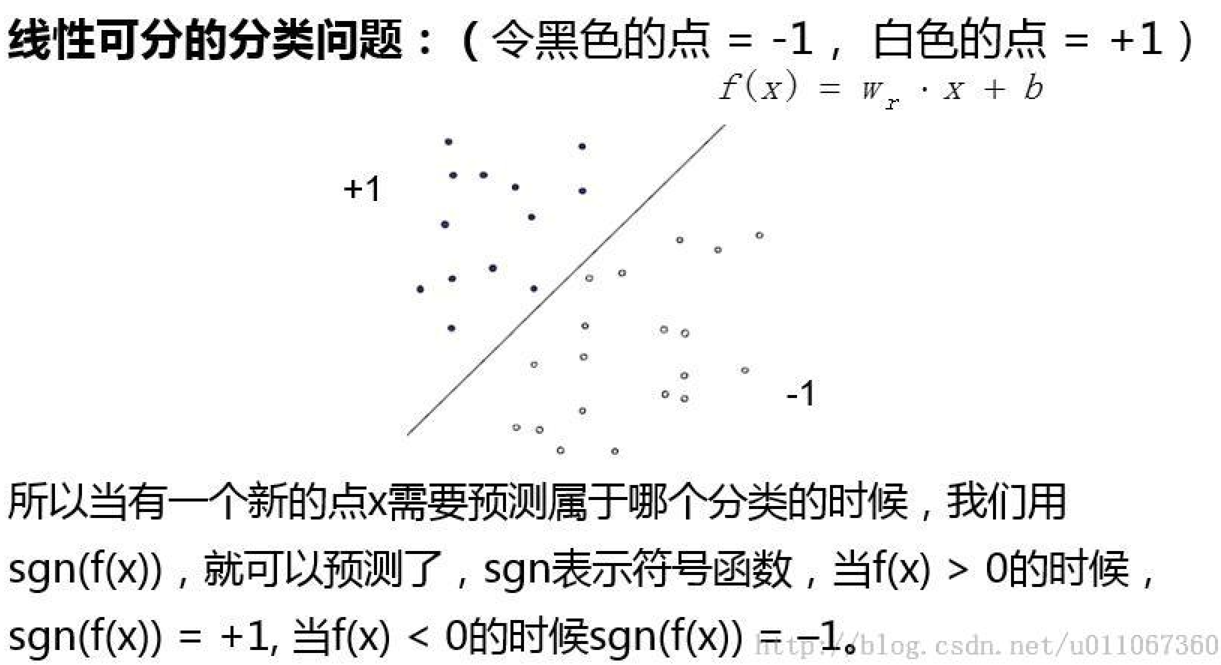
A、线性可分问题

1、**SVM**基本原理：

SVM使用一种非线性映射，把原训练数据映射到较高的维。在新的维上，搜索最佳分离超平面，两个类的数据总可以被超平面分开。



2、问题的提出：



3、如何选取最优的划分直线**f(x)**呢？

最大边缘超平面Maximum Marginal Hyperplane

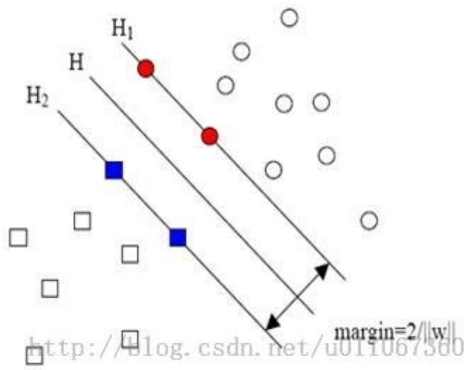
- 其中：W= {w1，w2，...wn} 是权重向量

H : WX+b=0　　H1:WX+b=1　　H2:WX+b=-1

- **几何间隔**：$\delta_{几何} = \frac{1}{\|w\|}|g(x)|$　　δ=y(wx+b)=|g(x)|
- **目标函数**　$\max \quad 2M = \frac{2}{\|w\|}$

等价于　$\min \quad \frac{1}{2}\|w\|^2$

- **约束条件**　$s.t. \begin{cases} w \cdot x_i + b \geq +1, & y_i = +1 \\ w \cdot x_i + b \leq -1, & y_i = -1 \end{cases}$



4、求解：凸二次规划

数据集合：$T = \{(x_1, y_1), \ldots, (x_l, y_l)\} \in (R^n \times y)^l$

$$x_i \in R^n, y_i \in Y = \{1, -1\}, i = 1, \ldots, l$$

目标函数：$\min \quad \frac{1}{2}\|w\|^2$

约束条件：$s.t. \begin{cases} w \cdot x_i + b \geq +1, & y_i = +1 \\ w \cdot x_i + b \leq -1, & y_i = -1 \end{cases}$

x，y已知　$\min \frac{1}{2}\|w\|^2 \quad s.t., y_i(w^T x_i + b) \geq 1, i = 1, \cdots, n$

建立拉格朗日函数：

$$L(w,b,a) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i (y_i(w^T x_i + b) - 1)$$

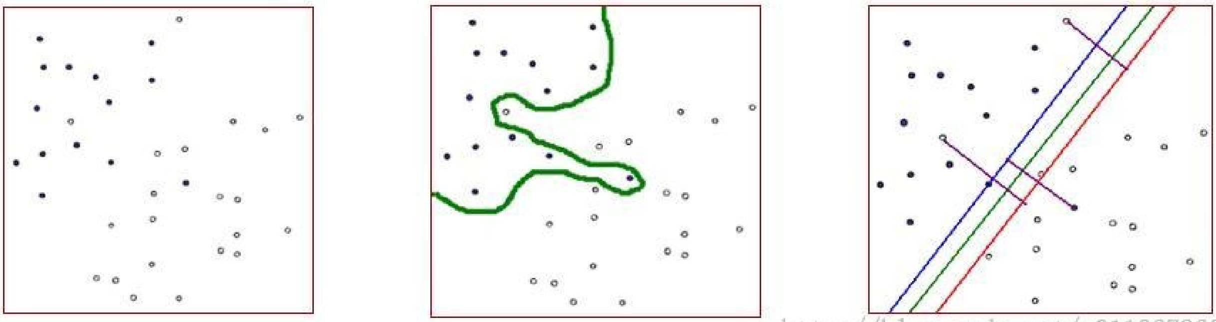
求偏导数：

$$\frac{\partial L}{\partial w}=0\Rightarrow w=\sum_{i=1}^{n}\alpha_i y_i x_i$$

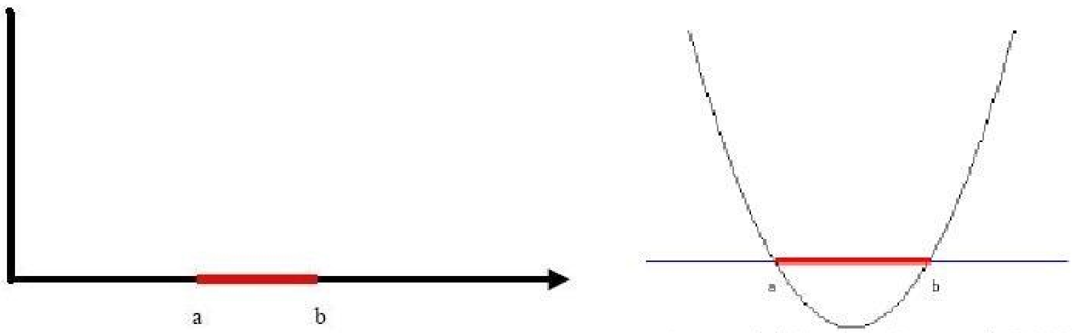
$$\frac{\partial L}{\partial w}=0\Rightarrow \sum_{i=1}^{n}\alpha_i y_i=0$$

B、线性不可分问题

1、核函数



如下图：横轴上端点a和b之间红色部分里的所有点定为正类，两边的黑色部分里的点定为负类.



设：

$$g(x)=c_0+c_1x+c_2x^2$$

g(x)转化为f(y)=

g(x)=f(y)=ay

在任意维度的空间中，这种形式的函数都是一个线性函数（只不过其中的a和y都是多维向量罢了），因为自变量y的次数不大于1。

$$y=\begin{bmatrix}y_1\\y_2\\y_3\end{bmatrix}=\begin{bmatrix}1\\x\\x^2\end{bmatrix},\quad a=\begin{bmatrix}a_1\\a_2\\a_3\end{bmatrix}=\begin{bmatrix}c_0\\c_1\\c_2\end{bmatrix}$$

下图w，x都是1000维，w'和x'分别是由w，x变换得到的2000维向量

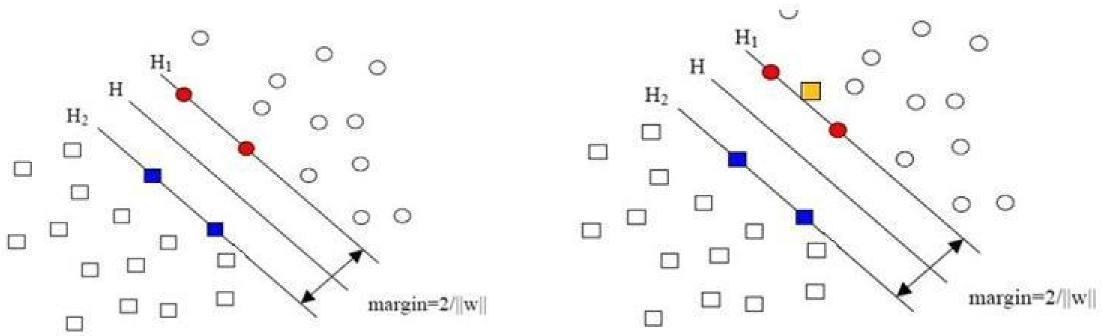
g(x)=K(w,x)+b

K(w,x)被称作核函数

基本作用：接受两个低维空间里的向量，能够计算出经过某个变换后在高维空间里的向量内积值。

$$f(x)=\langle w,x\rangle+b \quad f(x')=\langle w',x'\rangle+b$$

2、松弛变量



$$\min \quad \frac{1}{2}\|w\|^2+C\sum_{i=1}^{l}\zeta_i$$

$$\text{subject to} \quad y_i[(wx_i)+b]\geq 1-\zeta_i \quad (i=1,2,\ldots,l)$$

$$\zeta_i\geq 0$$

3、软间隔C-SVM：

$$\min \frac{1}{2}\|w\|^2$$
$$s.t., y_i(w^T x_i + b) \geq 1, i = 1, \cdots, n$$

$$\min \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{R}\varepsilon_i$$
$$s.t., y_i(w^T x_i + b) \geq 1 - \varepsilon_i, \varepsilon_i \geq 0$$

C是一个由用户指定的系数，表示对分错的点加入多少的惩罚，当C很大的时候，分错的点就会更少，但是过拟合的情况可能会比较严重，当C很小的时候，分错的点可能会很多，不过可能由此得到的模型也会不太正确

总结：

SVM算法优点：

(1)非线性映射是SVM方法的理论基础,SVM利用内积核函数代替向高维空间的非线性映射；

(2)对特征空间划分的最优超平面是SVM的目标,最大化分类边际的思想是SVM方法的核心；

(3)支持向量是SVM的训练结果,在SVM分类决策中起决定性作用。因此，模型需要存储空间小，算法鲁棒性（ Robust ）强。

SVM算法缺点：

(1) SVM算法对大规模训练样本难以实施

由于SVM是借助二次规划来求解支持向量，而求解二次规划将涉及m阶矩阵的计算（m为样本的个数），当m数目很大时该矩阵的存储和计算将耗费大量的机器内存和运算时间。

(2) 用SVM解决多分类问题存在困难

经典的支持向量机算法只给出了二类分类的算法，而在数据挖掘的实际应用中，一般要解决多类的分类问题。

基于以上问题，我们现在讨论SOM（Sequential Minimal Optimization algorithm）算法。

1、SMO算法的原理

这一被称为“顺次最小优化”的算法和以往的一些SVM改进算法一样，是把整个二次规划问题分解为很多易于处理的小问题，所不同的是，只有SMO算法把问题分解到可能达到的最小规模：每

次优化只处理两个样本的优化问题，并且用解析的方法进行处理。我们将会看到，这种与众不同的方法带来了一系列不可比拟的优势。

对SVM来说，一次至少要同时对两个样本进行优化（就是优化它们对应的Lagrange乘子），这是因为等式约束的存在使得我们不可能单独优化一个变量。

所谓“最小优化”的最大好处就是使得我们可以用解析的方法求解每一个最小规模的优化问题，从而完全避免了迭代算法。

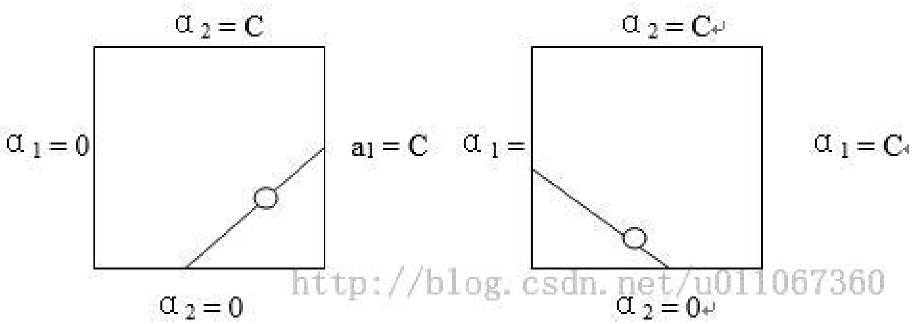
当然，这样一次“最小优化”不可能保证其结果就是所优化的Lagrange乘子的最终结果，但会使目标函数向极小值迈进一步。我们再对其它Lagrange乘子做最小优化，直到所有乘子都符合KKT

条件时，目标函数达到最小，算法结束。

这样，SMO算法要解决两个问题：一是怎样解决两个变量的优化问题，二是怎样决定先对哪些Lagrange乘子进行优化。

2、两个**Lagrange**乘子的优化问题

我们在这里不妨设正在优化的两个Lagrange乘子对应的样本正是第一个和第二个，对两个Lagrange乘子α_1和α_2，在其他乘子不改变的情况下，它们的约束条件应表达为正方形内的一条线段。（如图）



在这条线段上求一个函数的极值，相当于一个一维的极值问题。我们可以把α_1用α_2表示，对α_2求无条件极值，如果目标函数是严格上凹的，最小值就一定在这一极值点（极值点在区间内）或在区间端点（极值点在区间外）。α_2确定后，α_1也就确定下来了。因此我们先找到α_2优化区间的上下限制，再在这个区间中对α_2求最小值。

由图1我们容易得到α_2的上下限应为：

$L=\max(0,\alpha_2-\alpha_1)$，$H=\min(C,C+\alpha_2-\alpha_1)$, 若$y_1$与$y_2$异号；

$L=\max(0,\alpha_2+\alpha_1-C)$, $H=\min(C, \alpha_2+\alpha_1)$,若y_1与y_2同号；

令$s=y_1y_2$标志这两个样本是否同类，则有

$L=\max(0, \alpha_2+s\alpha_1-1/2\ (s+1)C)$, $H=\min(C, \alpha_2+s\alpha_1-1/2\ (s-1)C)$

而α_1和α_2在本次优化中所服从的等式约束为：

$\alpha_1+s\alpha_2=\alpha^0_1+s\alpha^0_2=d$

下面我们推导求最小值点α_2的公式：由于只有α_1，α_2两个变量需要考虑，目标函数可以写成

Wolfe(α_1,α_2)=1/2$K_{11}\alpha^2{}_1$+1/2 $K_{22}\alpha^2{}_2$+ s$K_{12}\alpha_1\alpha_2$ + $y_1\alpha_1v_1$+$y_2\alpha_2v_2$ －α_1 －α_2+常数

其中K_{ij}=K(x_i,x_j), v_i=$y_3\alpha^0{}_3K_{i3}$+…+$y_l\alpha^0{}_lK_{il}$ = u_i+b^0－ $y_1\alpha^0{}_1K_{1i}$ – $y_2\alpha^0{}_1K_{2i}$

上标为0的量表示是本次优化之前Lagrange乘子的原值。

将α_2用α_1表示并代入目标函数：

Wolfe（α_2）=1/2 K_{11}(d-sα_2)^2+1/2$K_{22}\alpha^2{}_2$+sK_{12}(d-sα_2) α_2+y_1(d-sα_2)v_1– d+sα_2+$y_2\alpha_2v_2$－α_2+常数

对α_2求导：

dWolfe(α_2)/dα_2=－sK_{11}(d－sα_2)+$K_{22}\alpha_2$－$K_{12}\alpha_2$+sK_{12}(d－sα_2)－y_2v_2+s+y_2v_2－1 ＝0

如果Wolfe函数总是严格上凹的，即二阶导数K_{11}+K_{22}－2K_{12}>0, 那么驻点必为极小值点，无条件的极值点就为

α_2＝[s(K_{11}－K_{12})d+y_2(v_1－v_2)＋1－s] / (K_{11}＋K_{22}－2K_{12})

将d,v与α^0,u之间关系代入，就得到用上一步的$\alpha^0{}_2$，u_1,u_2表示的α_2的无条件最优点：

α_2＝[$\alpha^0{}_2$(K_{11}＋K_{22}－2K_{12})+y_2(u_1－u_2＋y_2－y_1)] / (K_{11}＋K_{22}－2K_{12})

令η=K_{11}＋K_{22}－2K_{12}为目标函数的二阶导数，E_i=u_i－y_i为第i个训练样本的“误差”，这个式子又可以写为

α_2＝$\alpha^0{}_2$＋y_2(E_1－E_2)/η

除非核函数K不满足Mercer条件（也就是说不能作为核函数），η不会出现负值。但η=0是可以出现的情况。这时我们计算目标函数在线段两个端点上的取值，并将Lagrange乘子修正到目标函数较小的端点上：

f_1=y_1(E_1+b)－α_1K($\mathbf{x_1}$,$\mathbf{x_1}$)－sα_2K($\mathbf{x_1}$,$\mathbf{x_1}$)

f_2=y_2(E_2+b)－sα_1K($\mathbf{x_1}$,$\mathbf{x_2}$)－α_2K($\mathbf{x_2}$,$\mathbf{x_2}$)

L_1=α_1+s(α_2－L)

H_1=α_1+s(α_2－H)

WolfeL=L_1f_1+Lf_2+1/2 $L^2{}_1$K($\mathbf{x_1}$,$\mathbf{x_1}$)+1/2L^2K($\mathbf{x_2}$,$\mathbf{x_2}$)+sLL_1K($\mathbf{x_1}$,$\mathbf{x_2}$)

WolfeH=H_1f_1+Hf_2+1/2 $H^2{}_1$K($\mathbf{x_1}$,$\mathbf{x_1}$)+1/2H^2K($\mathbf{x_2}$,$\mathbf{x_2}$)+sHH_1K($\mathbf{x_1}$,$\mathbf{x_2}$)

当两个端点上取得相同的目标函数值时，目标函数在整条线段上的取值都会是一样的（因为它是上凹的），这时不必对α_1，α_2作出修正。

α_2的无条件极值确定后，再考虑上下限的限制，最终的α_2为

最后，由等式约束确定α_1：

$\alpha_1^*=\alpha_1+s(\alpha_2-\alpha_2^*)$

3、SMO算法

3.1、SMO算法的目的无非是找出一个函数f(x)，这个函数能让我们把输入的数据x进行分类

将一个凸二次规划问题转换成下列形式（KKT条件）

$$\frac{1}{2}w^T w+C\sum_{i=1}^{l}\xi_i \qquad \begin{cases} \alpha_i=0 \Leftrightarrow y_i u_i \geq 1 & (a) \\ 0<\alpha_i<C \Leftrightarrow y_i u_i=1 & (b) \\ \alpha_i=C \Leftrightarrow y_i u_i \leq 1 & (c) \end{cases}$$

这里的ai是拉格朗日乘子(问题通过拉格朗日乘法数来求解)

对于（a）的情况，表明ai是正常分类，在边界内部（我们知道正确分类的点yi*f(xi)>=0）

对于（b）的情况，表明了ai是支持向量，在边界上

对于（c）的情况，表明了ai是在两条边界之间 而最优解需要满足KKT条件，即满足（a）（b）（c）条件都满足

3.2、以下几种情况出现将会出现不满足：

yiui<=1但是ai<C则是不满足的,而原本ai=C

yiui>=1但是ai>0则是不满足的而原本ai=0

yiui=1但是ai=0或者ai=C则表明不满足的，而原本应该是0<ai<C

所以要找出不满足KKT的这些ai，并更新这些ai，但这些ai又受到另外一个约束，即

$$\sum_{i=1}^{l} a_i y_i=0$$

通过另一个方法，即同时更新ai和aj，满足以下等式

$$a_i^{new} y_i+a_j^{new} y_j=a_i^{old} y_i+a_j^{old} y_j=\text{常数}$$

就能保证和为0的约束。

利用上面的式子消去ai

得到一个关于单变量aj的一个凸二次规划问题，不考虑其约束0<=aj<=C,可以得其解为：

$$\alpha_{\mathrm{j}}^{new} = \alpha_{\mathrm{j}} + \frac{y_{\mathrm{j}}(E_{\mathrm{i}} - E_{j})}{\eta}$$

其中：

$$E_i = u_i - y_i, \eta = k(x_i, x_i) + k(x_j, x_j) - 2k(x_i, x_j)$$

aj表示旧值，然后考虑约束0<=aj<=C可得到a的解析解为：

$$\alpha_j^{new,clipped} = \begin{cases} H & if \ \ \alpha_j^{new} \geq H \\ \alpha_j^{new} & if \ \ L < \alpha_j^{new} < H \\ L & if \ \ \alpha_j^{new} \leq L \end{cases}$$

其中：

$$\begin{cases} L = \max(0, \alpha_j - \alpha_i), H = \max(C, C + \alpha_j - \alpha_i) & if \ \ y_i \neq y_j \\ L = \max(0, \alpha_j + \alpha_i - C), H = \max(C, \alpha_j - \alpha_i) & if \ \ y_i = y_j \end{cases}$$

输入是x，是一个数组，组中每一个值表示一个特征。 输出是A类还是B类。（正类还是负类）

$$f(x) = \sum_{j=1}^{n} \alpha_j y_j k(x_j, x) + b$$

4、SMO算法的特点和优势 SMO算法和以往流行的SVM优化算法如块算法、固定工作样本集法相比，既有共同点，又有自己的独特之处。

共同点在于它们都是把一个大的优化问题分解为很多小问题来处理。块算法在每一步中将新加入样本中违反KKT条件的样本与原有的支持向量一起组成小问题的样本集进行优化，优化完毕后

只保留其中的支持向量，再加进来新的样本进入下一步。固定工作样本集法是每一步只收集新加入样本中“最坏”的样本，并将原来保留的支持向量集中“较好”的替换出去，以保持样本集大小不

变。SMO则是把每一步的优化问题缩减到了最小，它可以看作是固定工作样本集法的一种特殊情况：把工作样本集的大小固定为2，并且每一步用两个新的Lagrange乘子替换原有的全部乘子。

SMO的最大特色在于它可以采用解析的方法而完全避免了二次规划数值解法的复杂迭代过程。这不但大大节省了计算时间，而且不会牵涉到迭代法造成的误差积累（其它一些算法中这种误差

积累带来了很大的麻烦）。理论上SMO的每一步最小优化都不会造成任何误差积累，而如果用双精度数计算，舍入误差几乎可以忽略，于是所有的误差只在于最后一遍检验时以多大的公差要

求所有Lagrange乘子满足KKT条件。可以说SMO算法在速度和精度两方面都得到了保证。

SMO在内存的节省上也颇具特色。我们看到，由于SMO不涉及二次规划数值解法，就不必将核函数矩阵整个存在内存里，而数值解法每步迭代都要拿这个矩阵作运算。（4000个样本的核函

数矩阵需要128M内存！）于是SMO使用的内存是与样本集大小成线性增长的，而不象以往的算法那样成平方增长。在我们的程序中SMO算法最多占用十几兆内存。SMO使得存储空间问题不

再是SVM应用中的另一个瓶颈。

SMO算法对线性支持向量机最为有效，对非线性则不能发挥出全部优势，这是因为线性情况下每次最小优化后的重置工作都是很简单的运算，而非线性时有一步加权求和，占用了主要的时

间。其他算法对线性和非线性区别不大，因为凡是涉及二次规划数值解的算法都把大量时间花在求数值解的运算中了。

当大多数Lagrange乘子都在边界上时，SMO算法的效果会更好。 尽管SMO的计算时间仍比训练集大小增长快得多，但比起其它方法来还是增长得慢一个等级。因此SMO较适合大数量的样本。

数据挖掘十大算法--Apriori算法

> 来源：http://blog.csdn.net/u011067360/article/details/24810415

一、**Apriori** 算法概述

Apriori 算法是一种最有影响力的挖掘布尔关联规则的频繁项集的 算法，它是由Rakesh Agrawal 和RamakrishnanSkrikant 提出的。它使用一种称作逐层搜索的迭代方法，k- 项集用于探索（k+1）- 项集。首先，找出频繁 1- 项集的集合。该集合记作L1。L1 用于找频繁2- 项集的集合 L2，而L2 用于找L2，如此下去，直到不能找到 k- 项集。每找一个 Lk 需要一次数据库扫描。为提高频繁项集逐层产生的效率，一种称作Apriori 性质的重 要性质 用于压缩搜索空间。其运行定理在于一是频繁项集的所有非空子集都必须也是频繁的，二是非频繁项集的所有父集都是非频繁的。

二、 问题的引入

购物篮分析：引发性例子

Question：哪组商品顾客可能会在一次购物时同时购买？

关联分析
Solutions：
1：经常同时购买的商品可以摆近一点，以便进一步刺激这些商品一起销售。
2：规划哪些附属商品可以降价销售，以便刺激主体商品的捆绑销售。

三、关联分析的基本概念

1、支持度

关联规则A->B的支持度support=P(AB)，指的是事件A和事件B同时发生的概率。

2、置信度

置信度confidence=P(B|A)=P(AB)/P(A),指的是发生事件A的基础上发生事件B的概率。比如说在规则Computer => antivirus_software , 其中 support=2%, confidence=60%中，就表示的意思是所有的商品交易中有2%的顾客同时买了电脑和杀毒软件，并且购买电脑的顾客中有60%也购买了杀毒软件。

3、k项集

如果事件A中包含k个元素，那么称这个事件A为k项集，并且事件A满足最小支持度阈值的事件称为频繁k项集。

4、由频繁项集产生强关联规则

1）K维数据项集LK是频繁项集的必要条件是它所有K-1维子项集也为频繁项集，记为LK-1

2）如果K维数据项集LK的任意一个K-1维子集Lk-1，不是频繁项集，则K维数据项集LK本身也不是最大数据项集。

3）Lk是K维频繁项集，如果所有K-1维频繁项集合Lk-1中包含LK的K-1维子项集的个数小于K，则Lk不可能是K维最大频繁数据项集。

4）同时满足最小支持度阀值和最小置信度阀值的规则称为强规则。

例如：用一个简单的例子说明。表1是顾客购买记录的数据库D，包含6个事务。项集I={网球拍,网球,运动鞋,羽毛球}。考虑关联规则：网球拍⇒网球，事务1,2,3,4,6包含网球拍，事务1,2,6同时包含网球拍和网球，支持度$support=\frac{3}{6}=0.5$，置信度$confident=\frac{3}{5}=0.6$。若给定最小支持度$\alpha=0.5$，最小置信度$\beta=0.6$，关联规则网球拍⇒网球是有趣的，认为购买网球拍和购买网球之间存在强关联。

四、**Apriori**算法的基本思想：

Apriori算法过程分为两个步骤：

第一步通过迭代，检索出事务数据库中的所有频繁项集，即支持度不低于用户设定的阈值的项集；

第二步利用频繁项集构造出满足用户最小信任度的规则。

具体做法就是：

首先找出频繁1-项集，记为L_1；然后利用L_1来产生候选项集C_2，对C_2中的项进行判定挖掘出L_2，即频繁2-项集；不断如此循环下去直到无法发现更多的频繁k-项集为止。每挖掘一层L_k就需要扫描整个数据库一遍。算法利用了一个性质：

Apriori 性质：任一频繁项集的所有非空子集也必须是频繁的。意思就是说，生成一个k-itemset的候选项时，如果这个候选项有子集不在(k-1)-itemset(已经确定是frequent的)中时，那么这个候选项就不用拿去和支持度判断了，直接删除。具体而言：

1） 连接步

为找出L_k（所有的频繁k项集的集合），通过将L_{k-1}（所有的频繁k-1项集的集合）与自身连接产生候选k项集的集合。候选集合记作C_k。设l_1和l_2是L_{k-1}中的成员。记$l_i[j]$表示l_i中的第j项。假设Apriori算法对事务或项集中的项按字典次序排序，即对于（k-1）项集l_i，l_i[1]i</sub>[2]<……….i</sub>[k-1]。将L_{k-1}与自身连接，如果(l_1[1]=l_2[1])&&(l_1[2]=l_2[2])&&……..&& (l_1[k-2]=l_2[k-2])&&(l_1[k-1]2</sub>[k-1])，那认为l_1和l_2是可连接。连接l_1和l_2 产生的结果是{l_1[1],l_1[2],……,l_1[k-1],l_2[k-1]}。

2） 剪枝步

C_K是L_K的超集，也就是说，C_K的成员可能是也可能不是频繁的。通过扫描所有的事务（交易），确定C_K中每个候选的计数，判断是否小于最小支持度计数，如果不是，则认为该候选是频繁的。为了压缩C_k,可以利用Apriori性质：任一频繁项集的所有非空子集也必须是频繁的，反之，如果某个候选的非空子集不是频繁的，那么该候选肯定不是频繁的，从而可以将其从C_K中删除。

五、实例说明

实例一：下面以图例的方式说明该算法的运行过程：假设有一个数据库D，其中有4个事务记录，分别表示为：

TID	Items
T1	I1,I3,I4
T2	I2,I3,I5
T3	I1,I2,I3,I5
T4	I2,I5

这里预定最小支持度minSupport=2,下面用图例说明算法运行的过程：

1、扫描D，对每个候选项进行支持度计数得到表C1:

项集	支持度计数
{I1}	2
{I2}	3
{I3}	3
{I4}	1
{I5}	3

2、比较候选项支持度计数与最小支持度minSupport，产生1维最大项目集L1：

项集	支持度计数
{I1}	2
{I2}	3
{I3}	3
{I5}	3

3、由L1产生候选项集C2：

项集
{I1,I2}
{I1,I3}
{I1,I5}
{I2,I3}
{I2,I5}
{I3,I5}

4、扫描D，对每个候选项集进行支持度计数:

项集	支持度计数
{I1,I2}	1
{I1,I3}	2
{I1,I5}	1
{I2,I3}	2
{I2,I5}	3
{I3,I5}	2

5、比较候选项支持度计数与最小支持度minSupport，产生2维最大项目集L2：

项集	支持度计数
{I1,I3}	2
{I2,I3}	2
{I2,I5}	3
{I3,I5}	2

6、由L2产生候选项集C3：

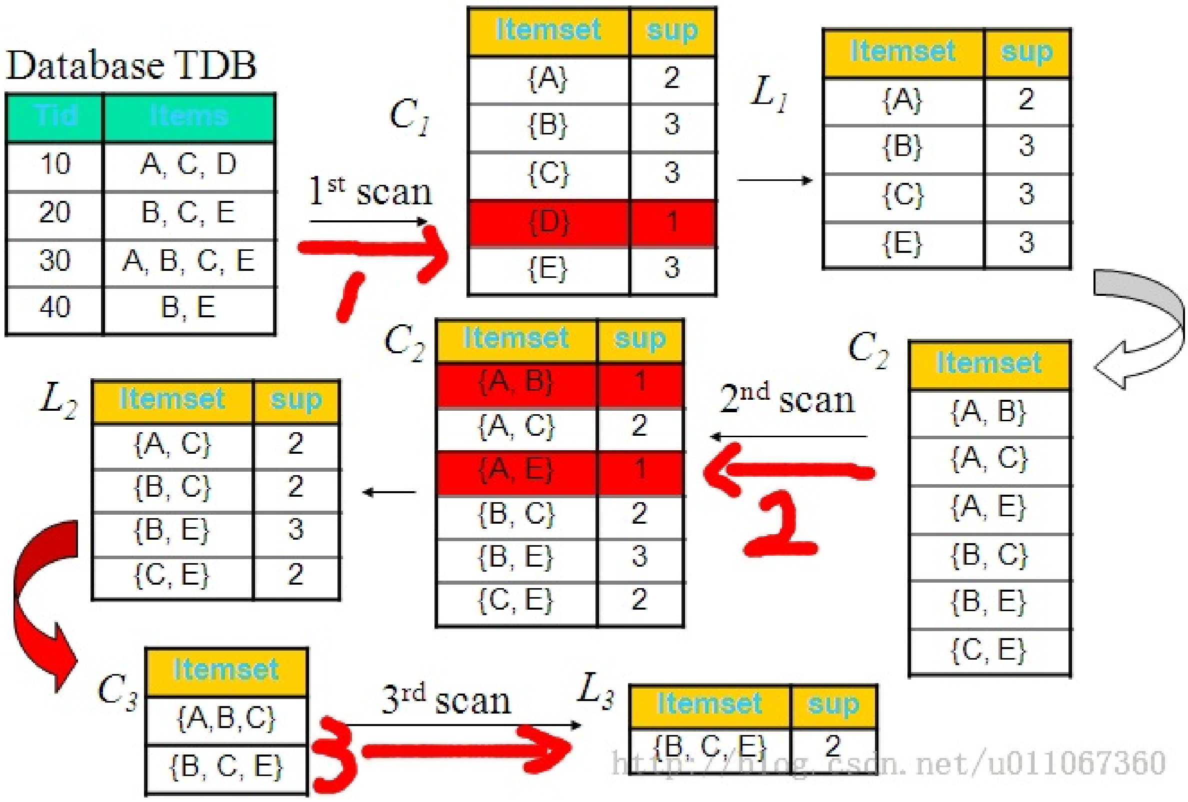
项集
{I2,I3,I5}

7、比较候选项支持度计数与最小支持度minSupport，产生3维最大项目集L3：

项集	支持度计数
{I2,I3,I5}	2

算法终止。

实例二：下图从整体同样的能说明此过程：



此例的分析如下：

1．连接：C3=L2 L2= {{A,C},{B,C},{B,E}{C,E}} {{A,C},{B,C},{B,E}{C,E}} = {{A,B,C},{A,C,E},{B,C,E}}

2. 使用Apriori性质剪枝：频繁项集的所有子集必须是频繁的，对候选项C3，我们可以删除其子集为非频繁的选项：

{A,B,C}的2项子集是{A,B},{A,C},{B,C}，其中{A,B}不是L2的元素，所以删除这个选项；

{A,C,E}的2项子集是{A,C},{A,E},{C,E}，其中{A,E} 不是L2的元素，所以删除这个选项；

{B,C,E}的2项子集是{B,C},{B,E},{C,E}，它的所有2－项子集都是L2的元素，因此保留这个选项。

3. 这样，剪枝后得到C3={{B,C,E}}

PS

从算法的运行过程，我们可以看出该Apriori算法的优点：简单、易理解、数据要求低，然而我们也可以看到Apriori算法的缺点：

(1)在每一步产生侯选项目集时循环产生的组合过多，没有排除不应该参与组合的元素;

(2)每次计算项集的支持度时，都对数据库D中的全部记录进行了一遍扫描比较，如果是一个大型的数据库的话，这种扫描比较会大大增加计算机系统的I/O开销。而这种代价是随着数据库的记录的增加呈现出几何级数的增加。因此人们开始寻求更好性能的算法。

六、改进**Apriori**算法的方法

方法1：基于hash表的项集计数

将每个项集通过相应的hash函数映射到hash表中的不同的桶中，这样可以通过将桶中的项集技术跟最小支持计数相比较先淘汰一部分项集。

方法2：事务压缩（压缩进一步迭代的事务数）

不包含任何k-项集的事务不可能包含任何(k+1)-项集，这种事务在下一步的计算中可以加上标记或删除

方法3：划分

挖掘频繁项集只需要两次数据扫描

D中的任何频繁项集必须作为局部频繁项集至少出现在一个部分中。

第一次扫描：将数据划分为多个部分并找到局部频繁项集

第二次扫描：评估每个候选项集的实际支持度，以确定全局频繁项集。

方法4：选样（在给定数据的一个子集挖掘）

基本思想：选择原始数据的一个样本，在这个样本上用Apriori算法挖掘频繁模式

通过牺牲精确度来减少算法开销，为了提高效率，样本大小应该以可以放在内存中为宜，可以适当降低最小支持度来减少遗漏的频繁模式

可以通过一次全局扫描来验证从样本中发现的模式

可以通过第二此全局扫描来找到遗漏的模式

方法5：动态项集计数

在扫描的不同点添加候选项集，这样，如果一个候选项集已经满足最少支持度，则在可以直接将它添加到频繁项集，而不必在这次扫描的以后对比中继续计算。

PS：Apriori算法的优化思路

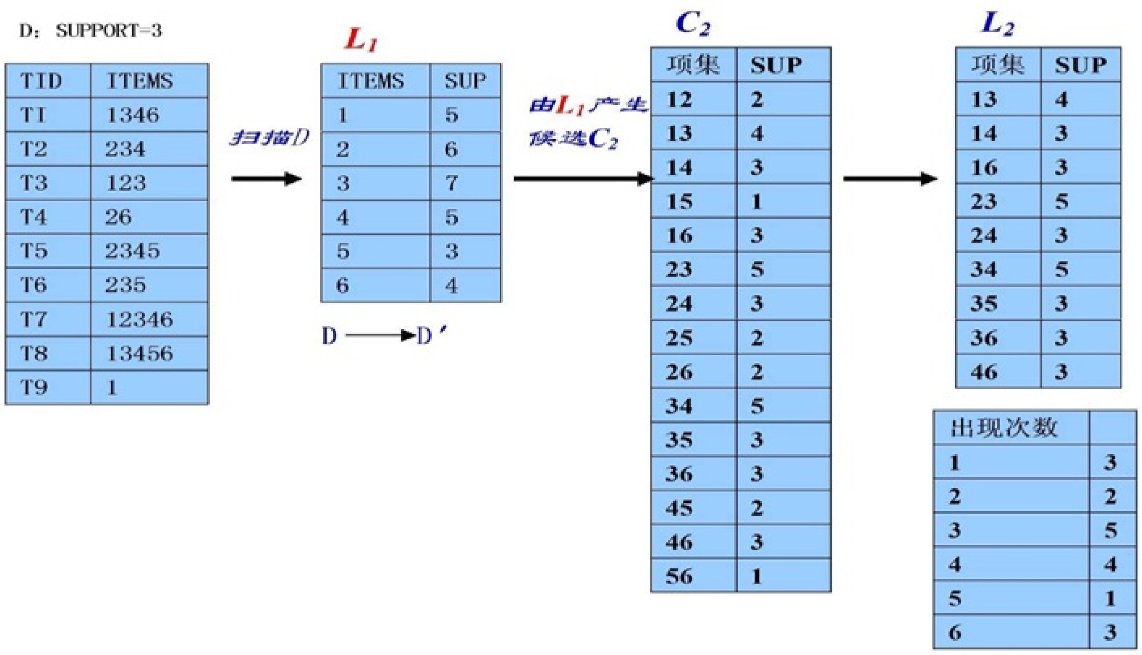
1、在逐层搜索循环过程的第k步中，根据k-1步生成的k-1维频繁项目集来产生k维候选项目集，由于在产生k-1维频繁项目集时，我们可以实现对该集中出现元素的个数进行计数处理，因此对某元素而言，若它的计数个数不到k-1的话，可以事先删除该元素，从而排除由该元素将引起的大规格所有组合。

这是因为对某一个元素要成为K维项目集的一元素的话，该元素在k-1阶频繁项目集中的计数次数必须达到K-1个，否则不可能生成K维项目集(性质3)。

2、根据以上思路得到了这个候选项目集后，可以对数据库D的每一个事务进行扫描，若该事务中至少含有候选项目集Ck中的一员则保留该项事务，否则把该事物记录与数据库末端没有作删除标记的事务记录对换，并对移到数据库末端的事务记录作删除标一记，整个数据库扫描完毕后为新的事务数据库D' 中。

因此随着K 的增大，D'中事务记录量大大地减少，对于下一次事务扫描可以大大节约I/0 开销。由于顾客一般可能一次只购买几件商品，因此这种虚拟删除的方法可以实现大量的交易记录在以后的挖掘中被剔除出来，在所剩余的不多的记录中再作更高维的数据挖掘是可以大大地节约时间的。

实例过程如下图：



TID	ITEMS
T1	1346
T2	234
T3	123
T4	26
T5	2345
T6	235
T7	12346
T8	13456
T9	1

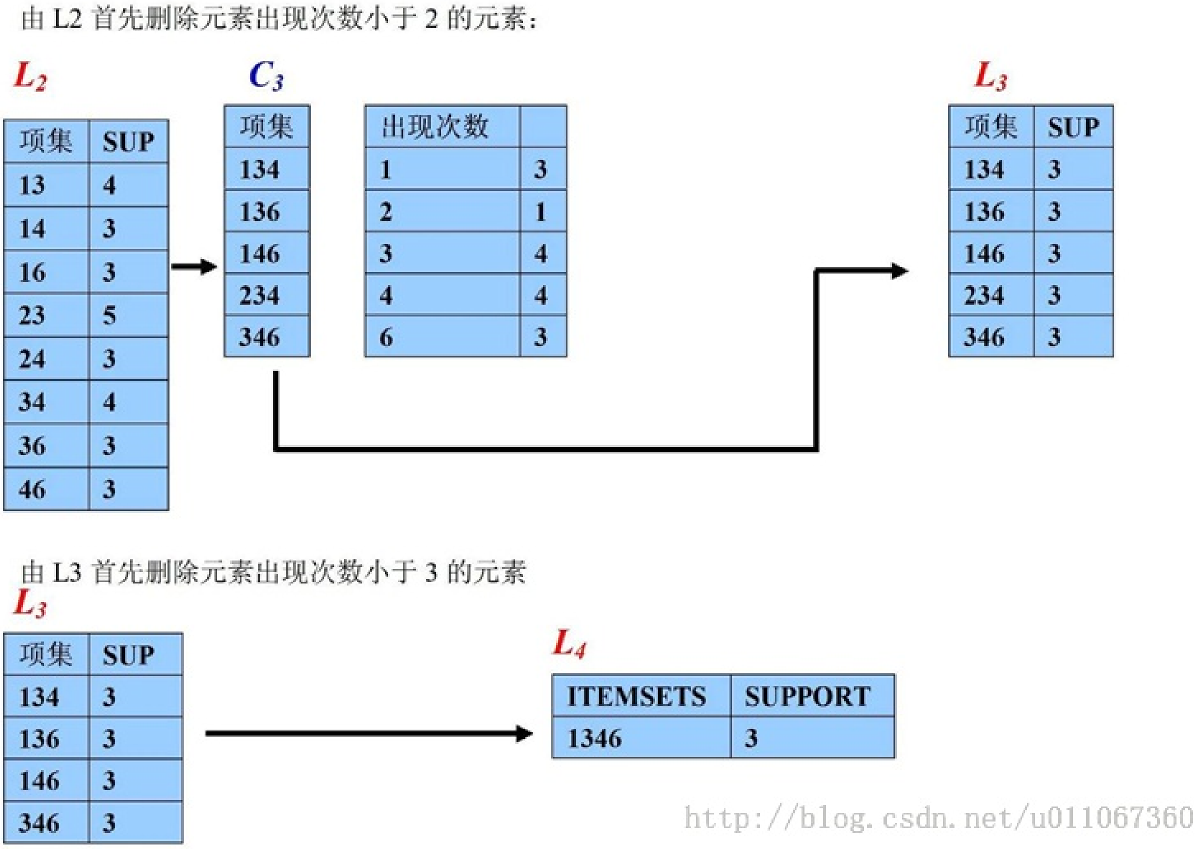
ITEMS	SUP
1	5
2	6
3	7
4	5
5	3
6	4

项集	SUP
12	2
13	4
14	3
15	1
16	3
23	5
24	3
25	2
26	2
34	5
35	3
36	3
45	2
46	3
56	1

项集	SUP
13	4
14	3
16	3
23	5
24	3
34	5
35	3
36	3
46	3

出现次数	
1	3
2	2
3	5
4	4
5	1
6	3

（由于 D 中的 T9 交易项是单独，因此剔除该象后生成数据库 D' 记 D—>D' ）



由 L2 首先删除元素出现次数小于 2 的元素：

L_2

项集	SUP
13	4
14	3
16	3
23	5
24	3
34	4
36	3
46	3

C_3

项集
134
136
146
234
346

出现次数	
1	3
2	1
3	4
4	4
6	3

L_3

项集	SUP
134	3
136	3
146	3
234	3
346	3

由 L3 首先删除元素出现次数小于 3 的元素

L_3

项集	SUP
134	3
136	3
146	3
346	3

L_4

ITEMSETS	SUPPORT
1346	3

当然还有很多相应的优化算法，比如针对Apriori算法的性能瓶颈问题-需要产生大量候选项集和需要重复地扫描数据库，2000年Jiawei Han等人提出了基于FP树生成频繁项集的FP-growth算法。该算法只进行2次数据库扫描且它不使用侯选集，直接压缩数据库成一个频繁模式树，最后通过这棵树生成关联规则。研究表明它比Apriori算法大约快一个数量级。

数据挖掘十大算法----EM算法（最大期望算法）

> 来源：http://blog.csdn.net/u011067360/article/details/23702125

概念

在统计计算中，最大期望（EM）算法是在概率（probabilistic）模型中寻找参数最大似然估计或者最大后验估计的算法，其中概率模型依赖于无法观测的隐藏变量（Latent Variable）。

最大期望经常用在机器学习和计算机视觉的数据聚类（Data Clustering）领域。

可以有一些比较形象的比喻说法把这个算法讲清楚。

比如说食堂的大师傅炒了一份菜，要等分成两份给两个人吃，显然没有必要拿来天平一点一点的精确的去称分量，最简单的办法是先随意的把菜分到两个碗中，然后观察是否一样多，把比较多的那一份取出一点放到另一个碗中，这个过程一直迭代地执行下去，直到大家看不出两个碗所容纳的菜有什么分量上的不同为止。（来自百度百科）

EM算法就是这样，假设我们估计知道A和B两个参数，在开始状态下二者都是未知的，并且知道了A的信息就可以得到B的信息，反过来知道了B也就得到了A。可以考虑首先赋予A某种初值，以此得到B的估计值，然后从B的当前值出发，重新估计A的取值，这个过程一直持续到收敛为止。

EM算法还是许多非监督聚类算法的基础（如Cheeseman et al. 1988），而且它是用于学习部分可观察马尔可夫模型（Partially Observable Markov Model）的广泛使用的Baum-Welch前向后向算法的基础。

估计*k*个高斯分布的均值

介绍EM算法最方便的方法是通过一个例子。

考虑数据*D*是一实例集合，它由*k*个不同正态分布的混合所得分布所生成。该问题框架在下图中示出，其中*k*=2而且实例为沿着*x*轴显示的点。

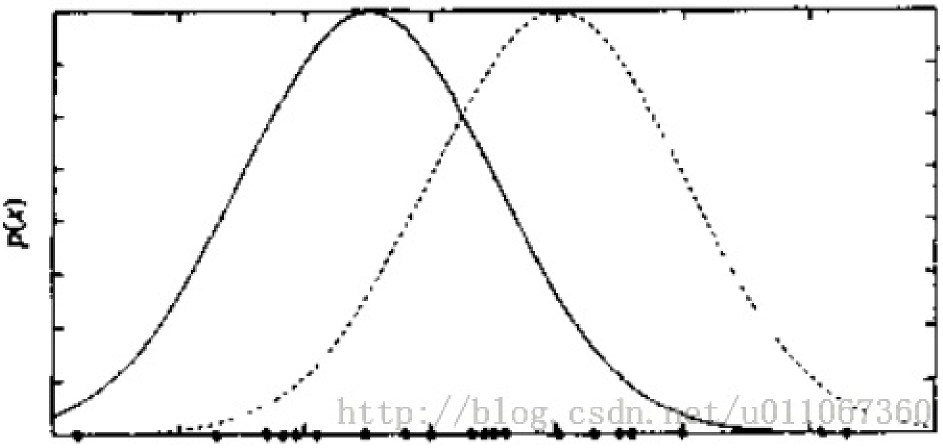
每个实例使用一个两步骤过程形成。

首先了随机选择*k*个正态分布其中之一。

其次随机变量x_i按照此选择的分布生成。

这一过程不断重复，生成一组数据点如图所示。为使讨论简单化，我们考虑一个简单情形，即单个正态分布的选择基于统一的概率进行选择，并且*k*个正态分布有相同的方差σ^2，且σ^2已知。

学习任务是输出一个假设$h=<\mu_1...\mu_k>$，它描述了k个分布中每一个分布的均值。我们希望对这些均值找到一个极大似然假设，即一个使$P(D|h)$最大化的假设h。



注意到，当给定从一个正态分布中抽取的数据实例$x_1,x_2, ..., x_m$时，很容易计算该分布的均值的极大似然假设。

其中我们可以证明极大似然假设是使m个训练实例上的误差平方和最小化的假设。

使用当表述一下式，可以得到：

$$\mu_{ML} = \frac{1}{m}\sum_{i=1}^{m} x_i$$

（公式一）

然而，在这里我们的问题涉及到k个不同正态分布的混合，而且我们不能知道哪个实例是哪个分布产生的。因此这是一个涉及隐藏变量的典型例子。

EM算法步骤

在上图的例子中，可把每个实例的完整描述看作是三元组$<x_i,z_{i1}, z_{i2}>$，其中x_i是第i个实例的观测值，z_{i1}和z_{i2}表示两个正态分布中哪个被用于产生值x_i。

确切地讲，z_{ij}在x_i由第j个正态分布产生时值为1，否则为0。这里x_i是实例的描述中已观察到的变量，z_{i1}和z_{i2}是隐藏变量。如果z_{i1}和z_{i2}的值可知，就可以用式一来解决均值μ_1和μ_2。因为它们未知，因此我们只能用EM算法。

EM算法应用于我们的k均值问题，目的是搜索一个极大似然假设，方法是根据当前假设$<\mu_1...\mu_k>$不断地再估计隐藏变量z_{ij}的期望值。然后用这些隐藏变量的期望值重新计算极大似然假设。这里首先描述这一实例化的EM算法，以后将给出EM算法的一般形式。

为了估计上图中的两个均值，EM算法首先将假设初始化为$h=<\mu_1,\mu_2>$，其中μ_1和μ_2为任意的初始值。然后重复以下的两个步骤以重估计h，直到该过程收敛到一个稳定的h值。

步骤1：计算每个隐藏变量z_{ij}的期望值$E[z_{ij}]$，假定当前假设$h=<\mu_1,\mu_2>$成立。

步骤2：计算一个新的极大似然假设$h'=<\mu_1',\mu_2'>$，假定由每个隐藏变量z_{ij}所取的值为第1步中得到的期望值$E[z_{ij}]$，然后将假设$h=<\mu_1,\mu_2>$替换为新的假设$h'=<\mu_1',\mu_2'>$，然后循环。

现在考察第一步是如何实现的。步骤1要计算每个z_{ij}的期望值。此$E[z_{ij}]$正是实例x_i由第j个正态分布生成的概率：

$$E[z_{ij}]=\frac{p(x=x_i\mid\mu=\mu_j)}{\sum_{n=1}^{2}p(x=x_i\mid\mu=\mu_n)}$$

$$=\frac{e^{-\frac{1}{2\sigma^2}(x_i-\mu_j)^2}}{\sum_{n=1}^{2}e^{-\frac{1}{2\sigma^2}(x_i-\mu_n)^2}}$$

因此第一步可由将当前值$<\mu_1,\mu_2>$和已知的x_i代入到上式中实现。

在第二步，使用第1步中得到的$E[z_{ij}]$来导出一新的极大似然假设$h'=<\mu_1',\mu_2'>$。如后面将讨论到的，这时的极大似然假设为：

$$\mu_j\leftarrow\frac{\sum_{i=1}^{m}E[z_{ij}]x_i}{\sum_{i=1}^{m}E[z_{ij}]}$$

注意此表达式类似于公式一中的样本均值，它用于从单个正态分布中估计μ。新的表达式只是对μ_j的加权样本均值，每个实例的权重为其由第j个正态分布产生的期望值。

上面估计$\boldsymbol{k}$个正态分布均值的算法描述了**EM**方法的要点：即当前的假设用于估计未知变量，而这些变量的期望值再被用于改进假设。

可以证明，在此算法第一次循环中，EM算法能使似然性$P(D|h)$增加，除非它已达到局部的最大。因此该算法收敛到对于$<\mu_1,\mu_2>$的一个局部极大可能性假设。

EM算法的一般表述

上面的EM算法针对的是估计混合正态分布均值的问题。更一般地，EM算法可用于许多问题框架，其中需要估计一组描述基准概率分布的参数θ，只给定了由此分布产生的全部数据中能观察到的一部分。

在上面的二均值问题中，感兴趣的参数为$\theta=<\mu_1,\mu_2>$，而全部数据为三元组$<x_i,z_{i1}, z_{i2}>$，而只有x_i可观察到，一般地令$X=<x_1, \ldots,x_m>$代表在同样的实例中已经观察到的数据，并令$Y=X\cup Z$代表全体数据。注意到未观察到的Z可被看作一随机变量，它的概率分布依赖于未知参数θ和已知数据X。类似地，Y是一随机变量，因为它是由随机变量Z来定义的。在后续部分，将描述EM算法的一般形式。使用h来代表参数θ的假设值，而h'代表在EM算法的每次迭代中修改的假设。

EM算法通过搜寻使$E[\ln P(Y|h')]$最大的h'来寻找极大似然假设h'。此期望值是在Y所遵循的概率分布上计算，此分布由未知参数θ确定。考虑此表达式究竟意味了什么。

首先$P(Y|h')$是给定假设h'下全部数据Y的似然性。其合理性在于我们要寻找一个h'使该量的某函数值最大化。

其次使该量的对数$\ln P(Y|h')$最大化也使$P(Y|h')$最大化，如已经介绍过的那样。

第三，引入期望值$E[\ln P(Y|h')]$是因为全部数据Y本身也是一随机变量。

已知全部数据Y是观察到的X和未观察到的Z的合并，我们必须在未观察到的Z的可能值上取平均，并以相应的概率为权值。换言之，要在随机变量Y遵循的概率分布上取期望值$E[\ln P(Y|h')]$。该分布由完全已知的X值加上Z服从的分布来确定。

Y遵从的概率分布是什么？一般来说不能知道此分布，因为它是由待估计的θ参数确定的。然而，EM算法使用其当前的假设h代替实际参数θ，以估计Y的分布。现定义一函数$Q(h'|h)$，它将$E[\ln P(Y|h')]$作为h'的一个函数给出，在$\theta=h$和全部数据Y的观察到的部分X的假定之下。

$$Q(h'|h) = E[\ln p(Y|h')|h, X]$$

将Q函数写成$Q(h'|h)$是为了表示其定义是在当前假设h等于θ的假定下。在EM算法的一般形式里，它重复以下两个步骤直至收敛。

步骤1：估计（E）步骤：使用当前假设h和观察到的数据X来估计Y上的概率分布以计算$Q(h'|h)$。

$$Q(h'|h) \leftarrow E[\ln P(Y|h')|h, X]$$

步骤2：最大化（M）步骤：将假设h替换为使Q函数最大化的假设h'：

$$h \leftarrow \arg\max_{h'} Q(h'|h)$$

当函数Q连续时，EM算法收敛到似然函数$P(Y|h')$的一个不动点。若此似然函数有单个的最大值时，EM算法可以收敛到这个对h'的全局的极大似然估计。否则，它只保证收敛到一个局部最大值。因此，EM与其他最优化方法有同样的局限性，如之前讨论的梯度下降，线性搜索和变形梯度等。

总结来说，EM算法就是通过迭代地最大化完整数据的对数似然函数的期望，来最大化不完整数据的对数似然函数。

PageRank

> 来源：http://baike.baidu.com/view/844648.htm

PageRank，网页排名，又称网页级别、Google左侧排名或佩奇排名，是一种由[1] 根据网页之间相互的超链接计算的技术，而作为网页排名的要素之一，以Google公司创办人拉里·佩奇（Larry Page）之姓来命名。Google用它来体现网页的相关性和重要性，在搜索引擎优化操作中是经常被用来评估网页优化的成效因素之一。Google的创始人拉里·佩奇和谢尔盖·布林于1998年在斯坦福大学发明了这项技术。

PageRank通过网络浩瀚的超链接关系来确定一个页面的等级。Google把从A页面到B页面的链接解释为A页面给B页面投票，Google根据投票来源（甚至来源的来源，即链接到A页面的页面）和投票目标的等级来决定新的等级。简单的说，一个高等级的页面可以使其他低等级页面的等级提升。

概念

PageRank是Google专有的算法，用于衡量特定网页相对于搜索引擎索引中的其他网页而言的重要程度。它由Larry Page 和 Sergey Brin在20世纪90年代后期发明。PageRank实现了将链接价值概念作为排名因素。

PageRank将对页面的链接看成投票，指示了重要性。

算法

PageRank让链接来"投票"

一个页面的“得票数”由所有链向它的页面的重要性来决定，到一个页面的超链接相当于对该页投一票。一个页面的PageRank是由所有链向它的页面（“链入页面”）的重要性经过递归算法得到的。一个有较多链入的页面会有较高的等级，相反如果一个页面没有任何链入页面，那么它没有等级。

2005年初，Google为网页链接推出一项新属性nofollow，使得网站管理员和网站作者可以做出一些Google不计票的链接，也就是说这些链接不算作"投票"。nofollow的设置可以抵制评论垃圾。

假设一个由4个页面组成的小团体：A，B，C和D。如果所有页面都链向A，那么A的PR（PageRank）值将是B，C及D的Pagerank总和。

$$PR(A) = PR(B) + PR(C) + PR(D)$$

继续假设B也有链接到C，并且D也有链接到包括A的3个页面。一个页面不能投票2次。所以B给每个页面半票。以同样的逻辑，D投出的票只有三分之一算到了A的PageRank上。

$$PR(A) = \frac{PR(B)}{2} + \frac{PR(C)}{1} + \frac{PR(D)}{3}$$

换句话说，根据链出总数平分一个页面的PR值。

$$PR(A) = \frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)}$$

最后，所有这些被换算为一个百分比再乘上一个系数。由于“没有向外链接的页面”传递出去的PageRank会是0，所以，Google通过数学系统给了每个页面一个最小值：

$$PR(A) = \left(\frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)} + \cdots\cdots \right) d + \frac{1-d}{N}$$

说明：在Sergey Brin和Lawrence Page的1998年原文中给每一个页面设定的最小值是1-d，而不是这里的(1-d)/N。 所以一个页面的PageRank是由其他页面的PageRank计算得到。Google不断的重复计算每个页面的PageRank。如果给每个页面一个随机PageRank值（非0），那么经过不断的重复计算，这些页面的PR值会趋向于稳定，也就是收敛的状态。这就是搜索引擎使用它的原因。

指标

Google工具条上的PageRank指标从0到10。它似乎是一个对数标度算法，细节未知。PageRank是Google的商标，其技术亦已经申请专利。

PageRank近似于一个用户，是指在Internet上随机地单击链接将会到达特定网页的可能性。通常，能够从更多地方到达的网页更为重要，因此具有更高的PageRank。每个到其他网页的链接，都增加了该网页的PageRank。具有较高PageRank的网页一般都是通过更多其他网页的链接而提高的。

为了查看站点PageRank，请安装GOOGLE工具条并启用PageRank特性，或者在firefox安装SearchStatus插件。但是请注意，GOOGLE所指示的PageRank是个缓冲值，通常是过时的。

更新频率

PageRank值每年只发布几次，有时就得使用过时信息，因此，PageRank并不是一个非常精确的度量。GOOGLE自己也似乎在使用更精确的值来进行排名。

在GOOGLE使用来构造搜索结果页面的采集算法中，PageRank只是其中的一个因素。有可能在特定查询下，具有较低PageRank的页面仍然能够排在具有较高PageRank的页面前面。PageRank也不一定是相关的，它使用链接来衡量整体受欢迎程度，而不是使用相关主题。

GOOGLE在计算搜索排名时也考虑链接的相关程度，因此PageRank不应该成为搜索引擎营销的唯一重点。构建相关链接，通常也自然会带来较高的PageRank。此外，为了提高PageRank而特意构建太多的不相关链接也有可能损害站点的排名，因为GOOGLE试图检测并对不相关链接降分，认为这种链接是用于提高排名得分的。

PageRank还被用户广泛认为是站点可靠的因素，因为用户倾向于相信带有较高值的站点更为著名或权威。当然，这就是PageRank所设计的目标。这个概念是GOOGLE所认可的，因此GOOGLE通过减少或清零PageRank来惩罚那些垃圾或不相关站点。

其它算法

GOOGLE PageRank并不是唯一的链接相关的排名算法，而是最为广泛使用的一种。其他算法还有：

一、Hilltop 算法

二、ExpertRank

三、HITS

四、TrustRank

数据挖掘算法学习（八）Adaboost算法

> 来源：http://blog.csdn.net/iemyxie/article/details/40423907

Adaboost是一种迭代算法，其核心思想是针对同一个训练集训练不同的分类器（弱分类器），然后把这些弱分类器集合起来，构成一个更强的最终分类器（强分类器）。Adaboost算法本身是通过改变数据分布来实现的，它根据每次训练集之中每个样本的分类是否正确，以及上次的总体分类的准确率，来确定每个样本的权值。将修改过权值的新数据集送给下层分类器进行训练，最后将每次得到的分类器最后融合起来，作为最后的决策分类器。

算法概述

1、先通过对N个训练样本的学习得到第一个弱分类器；2、将分错的样本和其他的新数据一起构成一个新的N个的训练样本，通过对这个样本的学习得到第二个弱分类器；3、将1和2都分错了的样本加上其他的新样本构成另一个新的N个的训练样本，通过对这个样本的学习得到第三个弱分类器 4、最终经过提升的强分类器。即某个数据被分为哪一类要由各分类器权值决定。

与**boosting**算法比较

1. 使用加权后选取的训练数据代替随机选取的训练样本，这样将训练的焦点集中在比较难分的训练数据样本上；　　　2. 将弱分类器联合起来，使用加权的投票机制代替平均投票机制。让分类效果好的弱分类器具有较大的权重，而分类效果差的分类器具有较小的权重。

与Boosting算法不同的是，AdaBoost算法不需要预先知道弱学习算法学习正确率的下限即弱分类器的误差，并且最后得到的强分类器的分类精度依赖于所有弱分类器的分类精度，这样可以深入挖掘弱分类器算法的能力。

算法步骤

1. 给定训练样本集S，其中X和Y分别对应于正例样本和负例样本；T为训练的最大循环次数；2. 初始化样本权重为1/n，即为训练样本的初始概率分布；　　　3. 第一次迭代：(1)训练样本的概率分布相当，训练弱分类器;(2)计算弱分类器的错误率;(3)选取合适阈值，使得误差最小；(4)更新样本权重；　　　经T次循环后，得到T个弱分类器，按更新的权重叠加，最终得到的强分类器。

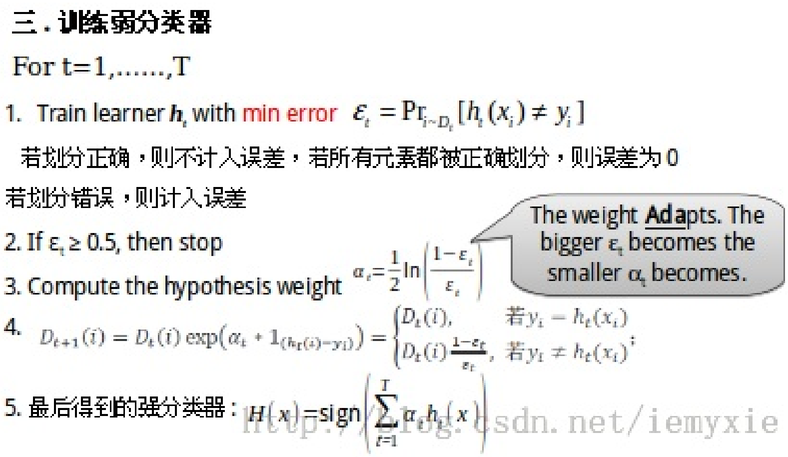
具体步骤如下：

一.样本

```
Given: m examples (x1, y1), …, (xm, ym)
     where xiX, yiY={-1, +1}
     xi表示X中第i个元素，
     yi表示与xi对应元素的属性值，+1表示xi属于某个分类，
     -1表示xi不属于某个分类
```

二.初始化训练样本

```
xi的权重D(i) :i=1,……, m;
     (1).若正负样本数目一致, D1(i) = 1/m
     (2).若正负样本数目m+, m-则正样本D1(i) = 1/m+,
         负样本D1(i) = 1/m-
```



三.训练弱分类器

For t=1,……,T

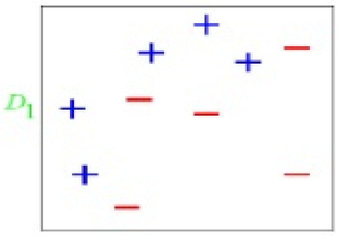
1. Train learner h_t with min error $\varepsilon_t = \Pr_{i\sim D_t}[h_t(x_i) \neq y_i]$

若划分正确，则不计入误差，若所有元素都被正确划分，则误差为0

若划分错误，则计入误差

2. If $\varepsilon_t \geq 0.5$, then stop
3. Compute the hypothesis weight $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
4. $D_{t+1}(i) = D_t(i)\exp(\alpha_t \cdot 1_{(h_t(i) - y_i)}) = \begin{cases} D_t(i), & 若 y_i = h_t(x_i) \\ D_t(i)\frac{1-\varepsilon_t}{\varepsilon_t}, & 若 y_i \neq h_t(x_i) \end{cases}$
5. 最后得到的强分类器：$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

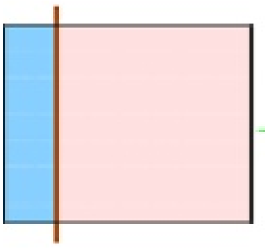
实例详解



图中"+"和"-"表示两种类别。我们用水平或者垂直的直线作为分类器进行分类。

算法开始前默认均匀分布D，共10个样本，故每个样本权值为0.1.

第一次分类：



第一次划分有3个点划分错误，根据误差表达式

$\varepsilon_t = \sum D_t \| y_{i \neq h_t(x_i)} \|$ 计算可得 `e1= (0.1+0.1+0.1) /1.0=0.3`

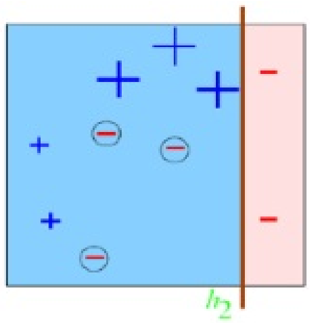
分类器权重：

$$\alpha_1 = \frac{1}{2}\ln\left(\frac{1-\varepsilon_1}{\varepsilon_1}\right) = \frac{1}{2}\ln\left(\frac{1-0.3}{0.3}\right) = 0.42$$

然后根据算法把错分点的权值变大。对于正确分类的7个点，权值不变，仍为0.1,对于错分的3个点，权值为：

```
D1=D0*(1-e1)/e1=0.1*(1-0.3)/0.3=0.2333
```

第二次分类：



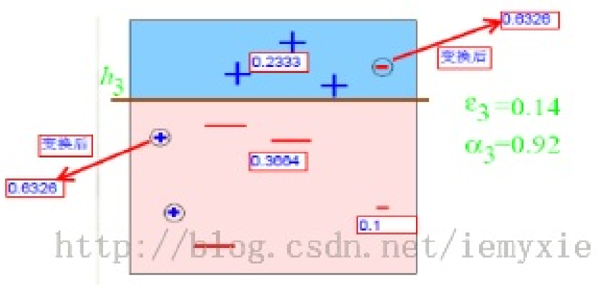
如图所示，有3个"-"分类错误。上轮分类后权值之和为：0.17+*0.23333*=1.3990

分类误差：`e2=0.1*3/1.3990=0.2144`

分类器权重 `a2=0.6493`

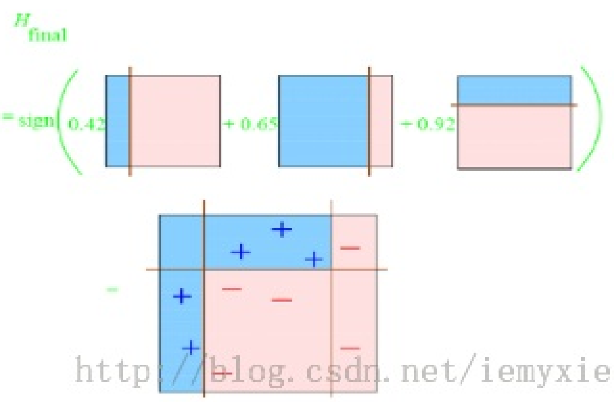
错分的3个点权值为：`D2=0.1*（1-0.2144）/0.2144=0.3664`

第三次分类：



同上步骤可求得：`e3=0.1365` ；`a3=0.9223` ；`D3=0.6326`

最终的强分类器即为三个弱分类器的叠加，如下图所示：



每个区域是属于哪个属性，由这个区域所在分类器的权值综合决定。比如左下角的区域，属于蓝色分类区的权重为h1 中的0.42和h2 中的0.65，其和为1.07；属于淡红色分类区域的权重为h3 中的0.92；属于淡红色分类区的权重小于属于蓝色分类区的权值，因此左下角属于蓝色分类区。因此可以得到整合的结果如上图所示，从结果图中看，即使是简单的分类器，组合起来也能获得很好的分类效果。

分类器权值调整的原因

$$\alpha_t=\frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$

由公式可以看到，权值是关于误差的表达式。每次迭代都会提高错分点的权值，当下一次分类器再次错分这些点之后，会提高整体的错误率，这样就导致分类器权值变小，进而导致这个分类器在最终的混合分类器中的权值变小，也就是说，Adaboost算法让正确率高的分类器占整体的权值更高，让正确率低的分类器权值更低，从而提高最终分类器的正确率。

算法优缺点

优点

1)Adaboost是一种有很高精度的分类器 2)可以使用各种方法构建子分类器,Adaboost算法提供的是框架 3)当使用简单分类器时，计算出的结果是可以理解的。而且弱分类器构造极其简单 4)简单，不用做特征筛选 5)不用担心overfitting(过度拟合)

缺点

1)容易受到噪声干扰，这也是大部分算法的缺点

2)训练时间过长

3)执行效果依赖于弱分类器的选择

SQL实现

```
#开始迭代
while(@i<=3) do
    set @evalue=0,@temp=0;
    set @flag1=0,@flag2=0,@flag3=0,@flag4=0;
    set @las=concat('d',@i-1);
    set @cur=concat('d',@i);
    set @a=concat('select hx,hy into @hx,@hy from hea where id = ',@i);
    prepare stmt1 from @a;
    execute stmt1;
    set @aa=concat('update adaset set ',@cur,' = ',@las);
    prepare stmt111 from @aa;
    execute stmt111;
#1.分类器为垂直x轴直线
    if (@hy=0) then
    #处理分类1
        set @b=concat('select count(class) into @l_1 from adaset where class=1 and x < ',
        prepare stmt2 from @b;
        execute stmt2;
        set @c=concat('select count(class) into @l_2 from adaset where class=-1 and x < '
        prepare stmt3 from @c;
        execute stmt3;
        if(@l_1=0 and @l_2!=0) then
```

```
        set @clas=concat('update hea set l=-1 where id = ',@i);
        prepare stmt28 from @clas;
        execute stmt28;
        end if;
        if(@l_1!=0 and @l_2 =0) then
            set @clas=concat('update hea set l=1 where id = ',@i);
            prepare stmt29 from @clas;
            execute stmt29;
        end if;
        set @weight=concat('d',@i-1);
        if (@l_1 !=0 and @l_2 !=0 and @l_1>@l_2) then #@l_2是错分点
            set @d=concat('select sum(',@weight,') into @temp from adaset where class=-1
            prepare stmt4 from @d;
            execute stmt4;
            set @evalue=@evalue+@temp;
            set @flag1=1;
            set @clas=concat('update hea set l=1 where id = ',@i);
            prepare stmt20 from @clas;
            execute stmt20;
        end if;
        if (@l_1 !=0 and @l_2 !=0 and @l_1<@l_2) then #@l_1是错分点
            set @d=concat('select sum(',@weight,') into @temp from adaset where class=1 a
            prepare stmt5 from @d;
            execute stmt5;
            set @evalue=@evalue+@temp;
            set @flag2=1;
            set @clas=concat('update hea set l=-1 where id = ',@i);
            prepare stmt21 from @clas;
            execute stmt21;
        end if;
#总权值&误差
    set @h=concat('select sum(',@weight,') into @temp from adaset');
    prepare stmt10 from @h;
    execute stmt10;
    set @evalue = round(@evalue/@temp,4);
    set @avalue = round((0.5*ln((1-@evalue)/@evalue)),4);
    set @eee=round((1-@evalue)/@evalue,4);
#更新误差e&假设权重a
    set @j=concat('update hea set e = ',@evalue,' ,a = ',@avalue,' where id = ',@i);
    prepare stmt11 from @j;
    execute stmt11;
#更新错分样本的权重
    if (@hy=0) then
        if (@flag1=1) then
            set @k=concat('update adaset set ',@cur,' = ',@las,'*',@eee,' where class=-1
            prepare stmt12 from @k;
            execute stmt12;
        end if;
        if (@flag2=1) then
            set @m=concat('update adaset set ',@cur,' = ',@las,'*',@eee,' where class=1 a
            prepare stmt13 from @m;
            execute stmt13;
        end if;
        if (@flag3=1) then
            set @n=concat('update adaset set ',@cur,' = ',@las,'*',@eee,' where class=-1
            prepare stmt14 from @n;
            execute stmt14;
        end if;
        if (@flag4=1) then
            set @o=concat('update adaset set ',@cur,' = ',@las,'*',@eee,' where class=1 a
            prepare stmt15 from @o;
            execute stmt15;
        end if;
    end if;
set @i=@i+1;
end while;
```

以上是博主最近用SQL实现的Adaboost算法的部分代码。数据库表以后整理一下再贴。Ubuntu不稳定啊，死机两次了。。编辑的博客都没了。。累觉不爱。。

个人疑问

上文中的缺点提到，Adaboost算法的效果依赖于弱分类器的选择，那么面对巨大的待分类数据时，如何选择弱分类呢？有没有什么原则。博主依旧在探索中，找到答案的话会在这里更新。

推荐资料：由Adaboost算法创始人Freund和Schapire写的关于Adaboost算法的文档，我已经上传。

数据挖掘十大算法--K近邻算法

> 来源：http://blog.csdn.net/u011067360/article/details/23941577

k-近邻算法是基于实例的学习方法中最基本的，先介绍基于实例学习的相关概念。

一、基于实例的学习。

1、已知一系列的训练样例，很多学习方法为目标函数建立起明确的一般化描述；但与此不同，基于实例的学习方法只是简单地把训练样例存储起来。

从这些实例中泛化的工作被推迟到必须分类新的实例时。每当学习器遇到一个新的查询实例，它分析这个新实例与以前存储的实例的关系，并据此把一个目标函数值赋给新实例。

2、基于实例的方法可以为不同的待分类查询实例建立不同的目标函数逼近。事实上，很多技术只建立目标函数的局部逼近，将其应用于与新查询实例邻近的实例，而从不建立在整个实例空间上都表现良好的逼近。当目标函数很复杂，但它可用不太复杂的局部逼近描述时，这样做有显著的优势。

3、基于实例方法的不足：

（1）分类新实例的开销可能很大。这是因为几乎所有的计算都发生在分类时，而不是在第一次遇到训练样例时。所以，如何有效地索引训练样例，以减少查询时所需计算是一个重要的实践问题。

（2）当从存储器中检索相似的训练样例时，它们一般考虑实例的所有属性。如果目标概念仅依赖于很多属性中的几个时，那么真正最“相似”的实例之间很可能相距甚远。

二、k-近邻法

基于实例的学习方法中最基本的是*k*-近邻算法。这个算法假定所有的实例对应于*n*维欧氏空间Â^{n}中的点。一个实例的最近邻是根据标准欧氏距离定义的。更精确地讲，把任意的实例*x*表示为下面的特征向量：

$a_1(x)$，$a_2(x)$，...，$a_n(x)$

其中$a_r(x)$表示实例*x*的第*r*个属性值。那么两个实例x_i和x_j间的距离定义为$d(x_i,x_j)$，其中：

$$d(x_i,x_j)\equiv\sqrt{\sum_{r=1}^{n}(a_r(x_i)-a_r(x_j))^2}$$

说明：

1、在最近邻学习中，目标函数值可以为离散值也可以为实值。

2、我们先考虑学习以下形式的离散目标函数$f: \Re^n \to V$。其中*V*是有限集合{v_1，...v_s}。下表给出了逼近离散目标函数的*k*-近邻算法。

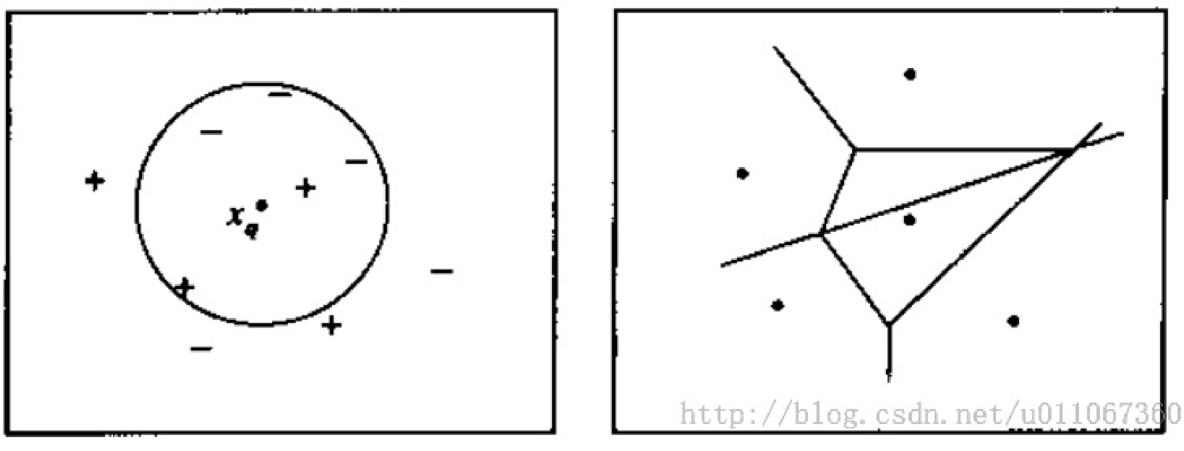
3、正如下表中所指出的，这个算法的返回值*f*′(x_q)为对*f*(x_q)的估计，它就是距离x_q最近的*k*个训练样例中最普遍的*f*值。

4、如果我们选择*k*=1，那么“1-近邻算法”就把*f*(x_i)赋给(x_q)，其中x_i是最靠近x_q的训练实例。对于较大的*k*值，这个算法返回前*k*个最靠近的训练实例中最普遍的*f*值。

逼近离散值函数*f*: Â^{n}</sup>V的_*k*-近邻算法

> 训练算法：
>
> 对于每个训练样例<_x_,_f_(_x_)>，把这个样例加入列表*training___examples*
>
> 分类算法：
>
> 给定一个要分类的查询实例x_q
>
> 在*training___examples*中选出最靠近x_q的*k*个实例，并用$x_1 .... x_k$表示
>
> 返回
>
> $$\hat{f}(x_q) \leftarrow \arg\max_{v \in V} \sum_{i=1}^{k} \delta(v, f(x_i))$$
>
> 其中如果*a*=*b*那么*d*(*a*,*b*)=1，否则*d*(*a*,*b*)=0。

下图图解了一种简单情况下的*k*-近邻算法，在这里实例是二维空间中的点，目标函数具有布尔值。正反训练样例用“+”和“-”分别表示。图中也画出了一个查询点x_q。注意在这幅图中，1-近邻算法把x_q分类为正例，然而5-近邻算法把x_q分类为反例。



图解说明：左图画出了一系列的正反训练样例和一个要分类的查询实例x_q。1-近邻算法把x_q分类为正例，然而5-近邻算法把x_q分类为反例。

右图是对于一个典型的训练样例集合1-近邻算法导致的决策面。围绕每个训练样例的凸多边形表示最靠近这个点的实例空间（即这个空间中的实例会被1-近邻算法赋予该训练样例所属的分类）。

对前面的*k*-近邻算法作简单的修改后，它就可被用于逼近连续值的目标函数。为了实现这一点，我们让算法计算*k*个最接近样例的平均值，而不是计算其中的最普遍的值。更精确地讲，为了逼近一个实值目标函数$f:\Re^n \rightarrow \Re$，我们只要把算法中的公式替换为：

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

三、距离加权最近邻算法

对*k*-近邻算法的一个显而易见的改进是对*k*个近邻的贡献加权，根据它们相对查询点x_q的距离，将较大的权值赋给较近的近邻。

例如，在上表逼近离散目标函数的算法中，我们可以根据每个近邻与x_q的距离平方的倒数加权这个近邻的“选举权”。

方法是通过用下式取代上表算法中的公式来实现：

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\arg\max} \sum_{i=1}^{k} w_i \delta(v, f(x_i))$$

其中

$$wi \equiv \frac{1}{d(x_q, x_i)^2}$$

为了处理查询点x_q恰好匹配某个训练样例x_i，从而导致分母为0的情况，我们令这种情况下的$f'(x_q)$等于$f(x_i)$。如果有多个这样的训练样例，我们使用它们中占多数的分类。

我们也可以用类似的方式对实值目标函数进行距离加权，只要用下式替换上表的公式：

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}$$

其中w_i的定义与之前公式中相同。

注意这个公式中的分母是一个常量，它将不同权值的贡献归一化（例如，它保证如果对所有的训练样例x_i，$f(x_i)=c$，那么(x_q)<--c）。

注意以上k-近邻算法的所有变体都只考虑k个近邻以分类查询点。如果使用按距离加权，那么允许所有的训练样例影响x_q的分类事实上没有坏处，因为非常远的实例对(x_q)的影响很小。考虑所有样例的惟一不足是会使分类运行得更慢。如果分类一个新的查询实例时考虑所有的训练样例，我们称此为全局（global）法。如果仅考虑最靠近的训练样例，我们称此为局部（local）法。

四、对***k***-近邻算法的说明

按距离加权的*k*-近邻算法是一种非常有效的归纳推理方法。它对训练数据中的噪声有很好的鲁棒性，而且当给定足够大的训练集合时它也非常有效。注意通过取*k*个近邻的加权平均，可以消除孤立的噪声样例的影响。

1、问题一：近邻间的距离会被大量的不相关属性所支配。

应用*k*-近邻算法的一个实践问题是，实例间的距离是根据实例的所有属性（也就是包含实例的欧氏空间的所有坐标轴）计算的。这与那些只选择全部实例属性的一个子集的方法不同，例如决策树学习系统。

比如这样一个问题：每个实例由20个属性描述，但在这些属性中仅有2个与它的分类是有关。在这种情况下，这两个相关属性的值一致的实例可能在这个20维的实例空间中相距很远。结果，依赖这20个属性的相似性度量会误导*k*-近邻算法的分类。近邻间的距离会被大量的不相关属性所支配。这种由于存在很多不相关属性所导致的难题，有时被称为维度灾难（curse of dimensionality）。最近邻方法对这个问题特别敏感。

2、解决方法：当计算两个实例间的距离时对每个属性加权。

这相当于按比例缩放欧氏空间中的坐标轴，缩短对应于不太相关属性的坐标轴，拉长对应于更相关的属性的坐标轴。每个坐标轴应伸展的数量可以通过交叉验证的方法自动决定。

3、问题二：应用*k*-近邻算法的另外一个实践问题是如何建立高效的索引。因为这个算法推迟所有的处理，直到接收到一个新的查询，所以处理每个新查询可能需要大量的计算。

4、解决方法：目前已经开发了很多方法用来对存储的训练样例进行索引，以便在增加一定存储开销情况下更高效地确定最近邻。一种索引方法是*kd*-tree（Bentley 1975；Friedman et al. 1977），它把实例存储在树的叶结点内，邻近的实例存储在同一个或附近的结点内。通过测试新查询x_q的选定属性，树的内部结点把查询x_q排列到相关的叶结点。

机器学习与数据挖掘-K最近邻(KNN)算法的实现（java和python版）

> 来源：http://blog.csdn.net/u011067360/article/details/45937327

KNN算法基础思想前面文章可以参考，这里主要讲解java和python的两种简单实现，也主要是理解简单的思想。

python版本：

这里实现一个手写识别算法，这里只简单识别0~9熟悉，在上篇文章中也展示了手写识别的应用，可以参考：[机器学习与数据挖掘-logistic回归及手写识别实例的实现]()

输入：每个手写数字已经事先处理成32*32的二进制文本，存储为txt文件。0～9每个数字都有10个训练样本，5个测试样本。训练样本集如下图：左边是文件目录，右边是其中一个文件打开显示的结果，看着像1，这里有0~9，每个数字都有是个样本来作为训练集。



第一步：将每个txt文本转化为一个向量，即3232的数组转化为11024的数组，这个1*1024的数组用机器学习的术语来说就是特征向量。

```
def img2vector(filename):
    returnVect = zeros((1,1024))
    fr = open(filename)
    for i in range(32):
        lineStr = fr.readline()
        for j in range(32):
            returnVect[0,32*i+j] = int(lineStr[j])
    return returnVect
```

第二步：训练样本中有1010个图片，可以合并成一个1001024的矩阵，每一行对应一个图片，也就是一个txt文档。

```
def handwritingClassTest():

    hwLabels = []
    trainingFileList = listdir('trainingDigits')
    print trainingFileList
    m = len(trainingFileList)
    trainingMat = zeros((m,1024))
    for i in range(m):
        fileNameStr = trainingFileList[i]
        fileStr = fileNameStr.split('.')[0]
        classNumStr = int(fileStr.split('_')[0])
        hwLabels.append(classNumStr)
        #print hwLabels
        #print fileNameStr
        trainingMat[i,:] = img2vector('trainingDigits/%s' % fileNameStr)
        #print trainingMat[i,:]
        #print len(trainingMat[i,:])

    testFileList = listdir('testDigits')
    errorCount = 0.0
    mTest = len(testFileList)
    for i in range(mTest):
        fileNameStr = testFileList[i]
        fileStr = fileNameStr.split('.')[0]
        classNumStr = int(fileStr.split('_')[0])
        vectorUnderTest = img2vector('testDigits/%s' % fileNameStr)
        classifierResult = classify0(vectorUnderTest, trainingMat, hwLabels, 3)
        print "the classifier came back with: %d, the real answer is: %d" % (classifierRe
        if (classifierResult != classNumStr): errorCount += 1.0
    print "\nthe total number of errors is: %d" % errorCount
    print "\nthe total error rate is: %f" % (errorCount/float(mTest))
```

第三步：测试样本中有105个图片，同样的，对于测试图片，将其转化为11024的向量，然后计算它与训练样本中各个图片的“距离”（这里两个向量的距离采用欧式距离），然后对距离排序，选出较小的前k个，因为这k个样本来自训练集，是已知其代表的数字的，所以被测试图片所代表的数字就可以确定为这k个中出现次数最多的那个数字。

```
def classify0(inX, dataSet, labels, k):
    dataSetSize = dataSet.shape[0]
    #tile(A,(m,n))
    print dataSet
    print "----------------"
    print tile(inX, (dataSetSize,1))
    print "----------------"
    diffMat = tile(inX, (dataSetSize,1)) - dataSet
    print diffMat
    sqDiffMat = diffMat**2
    sqDistances = sqDiffMat.sum(axis=1)
    distances = sqDistances**0.5
    sortedDistIndicies = distances.argsort()
    classCount={}
    for i in range(k):
        voteIlabel = labels[sortedDistIndicies[i]]
        classCount[voteIlabel] = classCount.get(voteIlabel,0) + 1
    sortedClassCount = sorted(classCount.iteritems(), key=operator.itemgetter(1), reverse
    return sortedClassCount[0][0]
```

全部实现代码：

```
#-*-coding:utf-8-*-
from numpy import *
import operator
from os import listdir

def classify0(inX, dataSet, labels, k):
    dataSetSize = dataSet.shape[0]
    #tile(A,(m,n))
    print dataSet
    print "----------------"
    print tile(inX, (dataSetSize,1))
    print "----------------"
    diffMat = tile(inX, (dataSetSize,1)) - dataSet
    print diffMat
    sqDiffMat = diffMat**2
    sqDistances = sqDiffMat.sum(axis=1)
    distances = sqDistances**0.5
    sortedDistIndicies = distances.argsort()
    classCount={}
    for i in range(k):
        voteIlabel = labels[sortedDistIndicies[i]]
        classCount[voteIlabel] = classCount.get(voteIlabel,0) + 1
    sortedClassCount = sorted(classCount.iteritems(), key=operator.itemgetter(1), reverse
    return sortedClassCount[0][0]

def img2vector(filename):
    returnVect = zeros((1,1024))
    fr = open(filename)
    for i in range(32):
        lineStr = fr.readline()
        for j in range(32):
            returnVect[0,32*i+j] = int(lineStr[j])
    return returnVect

def handwritingClassTest():

    hwLabels = []
    trainingFileList = listdir('trainingDigits')
    print trainingFileList
    m = len(trainingFileList)
    trainingMat = zeros((m,1024))
    for i in range(m):
        fileNameStr = trainingFileList[i]
        fileStr = fileNameStr.split('.')[0]
        classNumStr = int(fileStr.split('_')[0])
        hwLabels.append(classNumStr)
        #print hwLabels
        #print fileNameStr
        trainingMat[i,:] = img2vector('trainingDigits/%s' % fileNameStr)
        #print trainingMat[i,:]
        #print len(trainingMat[i,:])

    testFileList = listdir('testDigits')
    errorCount = 0.0
    mTest = len(testFileList)
    for i in range(mTest):
        fileNameStr = testFileList[i]
        fileStr = fileNameStr.split('.')[0]
        classNumStr = int(fileStr.split('_')[0])
        vectorUnderTest = img2vector('testDigits/%s' % fileNameStr)
        classifierResult = classify0(vectorUnderTest, trainingMat, hwLabels, 3)
        print "the classifier came back with: %d, the real answer is: %d" % (classifierRe
        if (classifierResult != classNumStr): errorCount += 1.0
    print "\nthe total number of errors is: %d" % errorCount
    print "\nthe total error rate is: %f" % (errorCount/float(mTest))

handwritingClassTest()
```

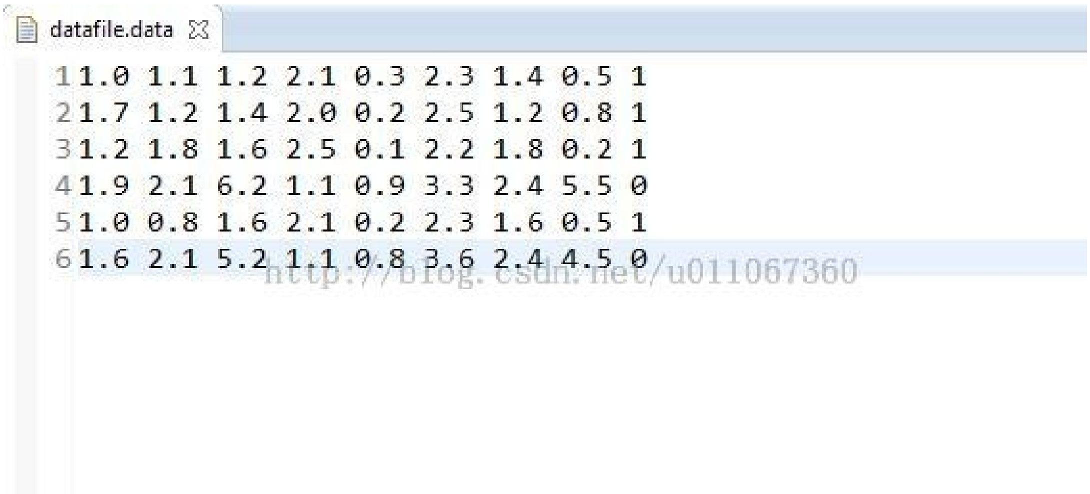
运行结果：源码文章尾可下载



java版本

先看看训练集和测试集：

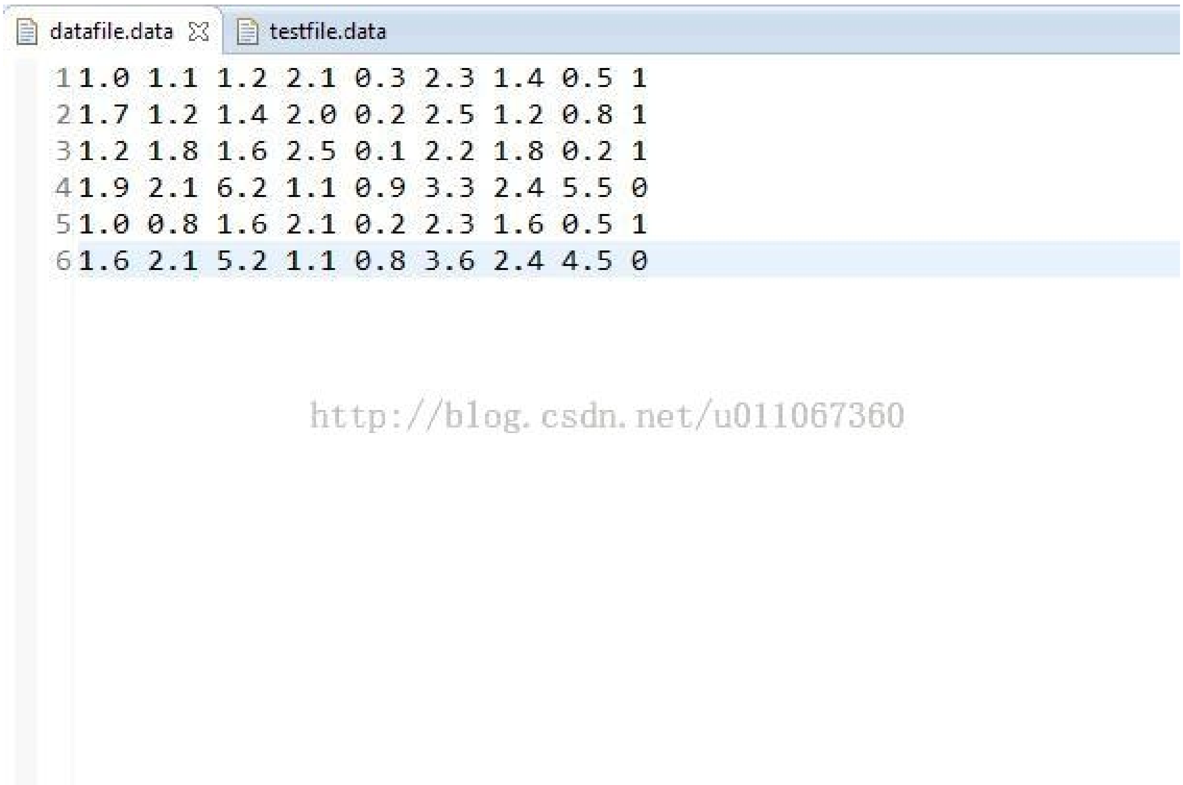
训练集：



datafile.data

```
1.0 1.1 1.2 2.1 0.3 2.3 1.4 0.5 1
1.7 1.2 1.4 2.0 0.2 2.5 1.2 0.8 1
1.2 1.8 1.6 2.5 0.1 2.2 1.8 0.2 1
1.9 2.1 6.2 1.1 0.9 3.3 2.4 5.5 0
1.0 0.8 1.6 2.1 0.2 2.3 1.6 0.5 1
1.6 2.1 5.2 1.1 0.8 3.6 2.4 4.5 0
```

测试集：




```
1.0 1.1 1.2 2.1 0.3 2.3 1.4 0.5 1
1.7 1.2 1.4 2.0 0.2 2.5 1.2 0.8 1
1.2 1.8 1.6 2.5 0.1 2.2 1.8 0.2 1
1.9 2.1 6.2 1.1 0.9 3.3 2.4 5.5 0
1.0 0.8 1.6 2.1 0.2 2.3 1.6 0.5 1
1.6 2.1 5.2 1.1 0.8 3.6 2.4 4.5 0
```


训练集最后一列代表分类（0或者1）

代码实现：

KNN算法主体类：

```
package Marchinglearning.knn2;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.PriorityQueue;

/**
 * KNN算法主体类
 */
public class KNN {
    /**
     * 设置优先级队列的比较函数，距离越大，优先级越高
     */
    private Comparator<KNNNode> comparator = new Comparator<KNNNode>() {
        public int compare(KNNNode o1, KNNNode o2) {
            if (o1.getDistance() >= o2.getDistance()) {
                return 1;
            } else {
                return 0;
            }
        }
    };
    /**
     * 获取K个不同的随机数
     * @param k 随机数的个数
     * @param max 随机数最大的范围
     * @return 生成的随机数数组
     */
    public List<Integer> getRandKNum(int k, int max) {
        List<Integer> rand = new ArrayList<Integer>(k);
```

```
        for (int i = 0; i < k; i++) {
            int temp = (int) (Math.random() * max);
            if (!rand.contains(temp)) {
                rand.add(temp);
            } else {
                i--;
            }
        }
        return rand;
    }
    /**
     * 计算测试元组与训练元组之前的距离
     * @param d1 测试元组
     * @param d2 训练元组
     * @return 距离值
     */
    public double calDistance(List<Double> d1, List<Double> d2) {
        System.out.println("d1:"+d1+",d2"+d2);
        double distance = 0.00;
        for (int i = 0; i < d1.size(); i++) {
            distance += (d1.get(i) - d2.get(i)) * (d1.get(i) - d2.get(i));
        }
        return distance;
    }
    /**
     * 执行KNN算法，获取测试元组的类别
     * @param datas 训练数据集
     * @param testData 测试元组
     * @param k 设定的K值
     * @return 测试元组的类别
     */
    public String knn(List<List<Double>> datas, List<Double> testData, int k) {
        PriorityQueue<KNNNode> pq = new PriorityQueue<KNNNode>(k, comparator);
        List<Integer> randNum = getRandKNum(k, datas.size());
        System.out.println("randNum:"+randNum.toString());
        for (int i = 0; i < k; i++) {
            int index = randNum.get(i);
            List<Double> currData = datas.get(index);
            String c = currData.get(currData.size() - 1).toString();
            System.out.println("currData:"+currData+",c:"+c+",testData"+testData);
            //计算测试元组与训练元组之前的距离
            KNNNode node = new KNNNode(index, calDistance(testData, currData), c);
            pq.add(node);
        }
        for (int i = 0; i < datas.size(); i++) {
            List<Double> t = datas.get(i);
            System.out.println("testData:"+testData);
            System.out.println("t:"+t);
            double distance = calDistance(testData, t);
            System.out.println("distance:"+distance);
            KNNNode top = pq.peek();
            if (top.getDistance() > distance) {
                pq.remove();
                pq.add(new KNNNode(i, distance, t.get(t.size() - 1).toString()));
            }
        }

        return getMostClass(pq);
    }
    /**
     * 获取所得到的k个最近邻元组的多数类
     * @param pq 存储k个最近近邻元组的优先级队列
     * @return 多数类的名称
     */
    private String getMostClass(PriorityQueue<KNNNode> pq) {
        Map<String, Integer> classCount = new HashMap<String, Integer>();
        for (int i = 0; i < pq.size(); i++) {
            KNNNode node = pq.remove();
            String c = node.getC();
            if (classCount.containsKey(c)) {
                classCount.put(c, classCount.get(c) + 1);
            } else {
```

```
                classCount.put(c, 1);
            }
        }
        int maxIndex = -1;
        int maxCount = 0;
        Object[] classes = classCount.keySet().toArray();
        for (int i = 0; i < classes.length; i++) {
            if (classCount.get(classes[i]) > maxCount) {
                maxIndex = i;
                maxCount = classCount.get(classes[i]);
            }
        }
        return classes[maxIndex].toString();
    }
}
```

KNN结点类，用来存储最近邻的k个元组相关的信息

```
package Marchinglearning.knn2;
/**
 * KNN结点类，用来存储最近邻的k个元组相关的信息
 */
public class KNNNode {
    private int index; // 元组标号
    private double distance; // 与测试元组的距离
    private String c; // 所属类别
    public KNNNode(int index, double distance, String c) {
        super();
        this.index = index;
        this.distance = distance;
        this.c = c;
    }

    public int getIndex() {
        return index;
    }
    public void setIndex(int index) {
        this.index = index;
    }
    public double getDistance() {
        return distance;
    }
    public void setDistance(double distance) {
        this.distance = distance;
    }
    public String getC() {
        return c;
    }
    public void setC(String c) {
        this.c = c;
    }
}
```

KNN算法测试类

```
package Marchinglearning.knn2;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.ArrayList;
import java.util.List;
/**
 * KNN算法测试类
 */
public class TestKNN {

    /**
     * 从数据文件中读取数据
     * @param datas 存储数据的集合对象
     * @param path 数据文件的路径
     */
    public void read(List<List<Double>> datas, String path){
        try {
            BufferedReader br = new BufferedReader(new FileReader(new File(path)));
            String data = br.readLine();
            List<Double> l = null;
            while (data != null) {
                String t[] = data.split(" ");
                l = new ArrayList<Double>();
                for (int i = 0; i < t.length; i++) {
                    l.add(Double.parseDouble(t[i]));
                }
                datas.add(l);
                data = br.readLine();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * 程序执行入口
     * @param args
     */
    public static void main(String[] args) {
        TestKNN t = new TestKNN();
        String datafile = new File("").getAbsolutePath() + File.separator +"knndata2"+Fil
        String testfile = new File("").getAbsolutePath() + File.separator +"knndata2"+Fil
        System.out.println("datafile:"+datafile);
        System.out.println("testfile:"+testfile);
        try {
            List<List<Double>> datas = new ArrayList<List<Double>>();
            List<List<Double>> testDatas = new ArrayList<List<Double>>();
            t.read(datas, datafile);
            t.read(testDatas, testfile);
            KNN knn = new KNN();
            for (int i = 0; i < testDatas.size(); i++) {
                List<Double> test = testDatas.get(i);
                System.out.print("测试元组: ");
                for (int j = 0; j < test.size(); j++) {
                    System.out.print(test.get(j) + " ");
                }
                System.out.print("类别为: ");
                System.out.println(Math.round(Float.parseFloat((knn.knn(datas, test, 3)))
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

运行结果为：

```
datafile:F:\androidproj\testjava\knndata2\datafile.data
testfile:F:\androidproj\testjava\knndata2\testfile.data
测试元组: 1.0 1.1 1.2 2.1 0.3 2.3 1.4 0.5 类别为: randNum:[1, 3, 5]
1
测试元组: 1.7 1.2 1.4 2.0 0.2 2.5 1.2 0.8 类别为: randNum:[3, 2, 4]
1
测试元组: 1.2 1.8 1.6 2.5 0.1 2.2 1.8 0.2 类别为: randNum:[5, 1, 0]
1
测试元组: 1.9 2.1 6.2 1.1 0.9 3.3 2.4 5.5 类别为: randNum:[2, 4, 1]
0
测试元组: 1.0 0.8 1.6 2.1 0.2 2.3 1.6 0.5 类别为: randNum:[0, 4, 1]
1
测试元组: 1.6 2.1 5.2 1.1 0.8 3.6 2.4 4.5 类别为: randNum:[1, 3, 4]
0
```


资源下载：

python版本下载

java版本下载

朴素贝叶斯分类器

> 来源：http://blog.csdn.net/u011067360/article/details/22890465

贝叶斯定理

贝叶斯定理解决了现实生活里经常遇到的问题：已知某条件概率，如何得到两个事件交换后的概率，也就是在已知P(A|B)的情况下如何求得P(B|A)。这里先解释什么是条件概率：

$P(A|B)$表示事件B已经发生的前提下，事件A发生的概率，叫做事件B发生下事件A的条件概率。其基本求解公式为：$P(A|B)=\frac{P(AB)}{P(B)}$。

贝叶斯定理之所以有用，是因为我们在生活中经常遇到这种情况：我们可以很容易直接得出P(A|B)，P(B|A)则很难直接得出，但我们更关心P(B|A)，贝叶斯定理就为我们打通从P(A|B)获得P(B|A)的道路。

下面不加证明地直接给出贝叶斯定理：

$$P(B|A)=\frac{P(A|B)P(B)}{P(A)}$$

朴素贝叶斯分类的原理与流程

朴素贝叶斯分类是一种十分简单的分类算法，叫它朴素贝叶斯分类是因为这种方法的思想真的很朴素。

朴素贝叶斯的思想基础是这样的：对于给出的待分类项，求解在此项出现的条件下各个类别出现的概率，哪个最大，就认为此待分类项属于哪个类别。

通俗来说，就好比这么个道理，你在街上看到一个黑人，我问你你猜这哥们哪里来的，你十有八九猜非洲。为什么呢？因为黑人中非洲人的比率最高，当然人家也可能是美洲人或亚洲人，但在没有其它可用信息下，我们会选择条件概率最大的类别，这就是朴素贝叶斯的思想基础。

朴素贝叶斯分类器应用的学习任务中，每个实例x可由属性值的合取描述，而目标函数$f(x)$从某有限集合V中取值。学习器被提供一系列关于目标函数的训练样例，以及新实例（描述为属性值的元组）$<a_1,a_2...a_n>$，然后要求预测新实例的目标值（或分类）。

贝叶斯方法的新实例分类目标是在给定描述实例的属性值$<a_1,a_2...a_n>$下，得到最可能的目标值V_{MAP}。

$$v_{MAP}=\arg\max_{v_j\in V} P(v_j\mid a_1,a_2...a_n)$$

可使用贝叶斯公式将此表达式重写为

$$v_{MAP}=\arg\max_{v_j\in V}\frac{P(a_1,a_2...a_n\mid v_j)P(v_j)}{P(a_1,a_2...a_n)}=\arg\max_{v_j\in V} P(a_1,a_2...a_n\mid v_j)P(v_j)$$

（1）

现在要做的是基于训练数据估计（1）式中两个数据项的值。估计每个$P(v_j)$很容易，只要计算每个目标值v_j出现在训练数据中的频率就可以。

然而，除非有一非常大的训练数据的集合，否则用这样方法估计不同的$P(a_1,a_2...a_n|v_j)$项不太可行。

问题在于这些项的数量等于可能实例的数量乘以可能目标值的数量。因此为获得合理的估计，实例空间中每个实例必须出现多次。

朴素贝叶斯分类器基于一个简单的假定：在给定目标值时属性值之间相互条件独立。

换言之，该假定说明给定实例的目标值情况下，观察到联合的$a_1,a_2...a_n$的概率正好是对每个单独属性的概率乘积：

$$P(a_1,a_2...a_n\mid v_j)=\prod_i P(a_i\mid v_j)$$

将其代入（1）式中，可得到朴素贝叶斯分类器所使用的方法：

朴素贝叶斯分类器：

$$v_{NB}=\arg\max_{v_j\in V} P(v_j)\prod_i P(a_i\mid v_j)$$

其中v_{NB}表示朴素贝叶斯分类器输出的目标值。

注意在朴素贝叶斯分类器中，须从训练数据中估计的不同$P(a_i|v_j)$项的数量只是不同的属性值数量乘以不同目标值数量——这比要估计$P(a_1,a_2...a_n|v_j)$项所需的量小得多。

概括地讲，朴素贝叶斯学习方法需要估计不同的$P(v_j)$和$P(a_i|v_j)$项，基于它们在训练数据上的频率。这些估计对应了待学习的假设。然后该假设使用上面式中的规则来分类新实例。只要所需的条件独立性能够被满足，朴素贝叶斯分类v_{NB}等于*MAP*分类。

朴素贝叶斯学习方法和其他已介绍的学习方法之间有一有趣的差别：没有明确的搜索假设空间的过程（这里，可能假设的空间为可被赋予不同的$P(v_j)$和$P(a_i|v_j)$项的可能值。相反，假设的形成不需要搜索，只是简单地计算训练样例中不同数据组合的出现频率）。

朴素贝叶斯分类的正式定义如下：

1、设$x=\{a_1,a_2,...,a_m\}$为一个待分类项，而每个a为x的一个特征属性。

2、有类别集合$C=\{y_1,y_2,...,y_n\}$。

3、计算$P(y_1|x),P(y_2|x),...,P(y_n|x)$。

4、如果$P(y_k|x)=max\{P(y_1|x),P(y_2|x),...,P(y_n|x)\}$，则$x\in y_k$。

那么现在的关键就是如何计算第3步中的各个条件概率。我们可以这么做：

1、找到一个已知分类的待分类项集合，这个集合叫做训练样本集。

2、统计得到在各类别下各个特征属性的条件概率估计。即
$P(a_1|y_1),P(a_2|y_1),...,P(a_m|y_1);P(a_1|y_2),P(a_2|y_2),...,P(a_m|y_2);...;P(a_1|y_n),P(a_2|y_n),...,P(a_m|y_n)$
。

3、如果各个特征属性是条件独立的，则根据贝叶斯定理有如下推导：

$$P(y_i|x)=\frac{P(x|y_i)P(y_i)}{P(x)}$$

因为分母对于所有类别为常数，因为我们只要将分子最大化皆可。又因为各特征属性是条件独立的，所以有：

$$P(x|y_i)P(y_i)=P(a_1|y_i)P(a_2|y_i)...P(a_m|y_i)P(y_i)=P(y_i)\prod_{j=1}^{m}P(a_j|y_i)$$

朴素贝叶斯分类实例：按照某人是否要打网球来划分天气

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

这里我们使用此表中的数据结合朴素贝叶斯分类器来分类下面的新实例：

```
Outlook=sunny, Temperature=cool,Humidity=high,Wind=strong
```

我们的任务是对此新实例预测目标概念*PlayTennis* 的目标值（*yes* 或*no*）。将上面式子应用到当前的任务，目标值v_{NB} 由下式给出：

$$v_{NB} = \arg\max_{v_j \in \{yes,no\}} P(v_j) \prod_i P(a_i \mid v_j)$$

$$= \arg\max_{v_j \in \{yes,no\}} P(v_j) P(Outlook = sunny \mid v_j) P(Temperature = cool \mid v_j)$$

$$P(Humidity = high \mid v_j) P(Wind = strong \mid v_j) \quad (2)$$

注意在最后一个表达式中a_i已经用新实例的特定属性值实例化了。为计算v_{NB}，现在需要10个概率，它们都可以训练数据中估计出。

首先不同目标值的概率可以基于这14个训练样例的频率很容易地估计出：

```
P(PlayTennis=yes)=9/14=0.64
P(PlayTennis=no)=5/14=0.36
```

相似地，可以估计出条件概率，例如对于*Wind=Strong*有：

```
P(Wind=strong|PlayTennis=yes)=3/9=0.33
P(Wind=strong|PlayTennis=no)=3/5=0.60
```

使用这些概率估计以及相似的对剩余属性的估计，可按照式（2）计算v_{NB}如下（为简明起见忽略了属性名）。

```
P(yes)P(sunny|yes)P(cool|yes)P(high|yes)P(strong|yes)=0.0053
P(no)P(sunny|no)P(cool|no)P(high|no)P(strong|no)=0.0206
```

这样，基于从训练数据中学习到的概率估计，朴素贝叶斯分类器将此实例赋以目标值*PlayTennis=no* 。

更进一步，通过将上述的量归一化，可计算给定观察值下目标值为*no* 的条件概率。对于此例，概率为0.0206/(0.0206+0.0053)=0.795。

从数学角度来说，分类问题可做如下定义：

已知集合：$C = \{y_1, y_2, ..., y_n\}$和$I = \{x_1, x_2, ..., x_m, \cdots\}$，确定映射规则$y = f(x)$，使得任意$x_i \in I$有且仅有一个$y_j \in C$使得$y_j = f(x_i)$成立。（不考虑模糊数学里的模糊集情况）

其中C叫做类别集合，其中每一个元素是一个类别，而I叫做项集合，其中每一个元素是一个待分类项，f叫做分类器。分类算法的任务就是构造分类器f。

这里要着重强调，分类问题往往采用经验性方法构造映射规则，即一般情况下的分类问题缺少足够的信息来构造100%正确的映射规则，而是通过对经验数据的学习从而实现一定概率意义上正确的分类，因此所训练出的分类器并不是一定能将每个待分类项准确映射到其分类，分类器的质量与分类器构造方法、待分类数据的特性以及训练样本数量等诸多因素有关。

数据挖掘十大经典算法--CART: 分类与回归树

> 来源：http://blog.csdn.net/u011067360/article/details/24871801

一、决策树的类型 在数据挖掘中，决策树主要有两种类型:

分类树 的输出是样本的类标。 回归树 的输出是一个实数 (例如房子的价格，病人呆在医院的时间等)。

术语分类和回归树 (CART) 包含了上述两种决策树, 最先由Breiman 等提出.分类树和回归树有些共同点和不同点—例如处理在何处分裂的问题。

分类回归树(CART,Classification And Regression Tree)也属于一种决策树，之前我们介绍了基于ID3和C4.5算法的决策树。这里只介绍CART是怎样用于分类的。

分类回归树是一棵二叉树，且每个非叶子节点都有两个孩子，所以对于第一棵子树其叶子节点数比非叶子节点数多1。

CART与ID3区别： CART中用于选择变量的不纯性度量是Gini指数； 如果目标变量是标称的，并且是具有两个以上的类别，则CART可能考虑将目标类别合并成两个超类别（双化）；如果目标变量是连续的，则CART算法找出一组基于树的回归方程来预测目标变量。

二、构建决策树

构建决策树时通常采用自上而下的方法，在每一步选择一个最好的属性来分裂。 "最好" 的定义是使得子节点中的训练集尽量的纯。不同的算法使用不同的指标来定义"最好"。本部分介绍一中最常见的指标。

有4中不同的不纯度量可以用来发现CART模型的划分，取决于目标变量的类型，对于分类的目标变量，可以选择GINI，双化或有序双化； 对于连续的目标变量，可以使用最小二乘偏差（LSD）或最小绝对偏差（LAD）。

下面我们只讲GINI指数。 GINI指数： 1、是一种不等性度量； 2、通常用来度量收入不平衡，可以用来度量任何不均匀分布； 3、是介于0~1之间的数，0-完全相等，1-完全不相等； 4、总体内包含的类别越杂乱，GINI指数就越大（跟熵的概念很相似）

CART分析步骤

1、从根节点t=1开始，从所有可能候选S集合中搜索使不纯性降低最大的划分S，然后，使用划分*S*将节点1（t=1）划分成两个节点t=2和t=3； 2、在t=2和t=3上分别重复划分搜索过程。

基尼不纯度指标 在CART算法中, 基尼不纯度表示一个随机选中的样本在子集中被分错的可能性。基尼不纯度为这个样本被选中的概率乘以它被分错的概率。当一个节点中所有样本都是一个类时，基尼不纯度为零。

假设y的可能取值为{1, 2, ..., m},令fi是样本被赋予i的概率，则基尼指数可以通过如下计算：

$$I_G(f)=\sum_{i=1}^{m}f_i(1-f_i)=\sum_{i=1}^{m}(f_i-{f_i}^2)=\sum_{i=1}^{m}f_i-\sum_{i=1}^{m}{f_i}^2=1-\sum_{i=1}^{m}{f_i}^2$$

例如：

名称	体温	表面覆盖	胎生	产蛋	能飞	水生	有腿	冬眠	类标记
人	恒温	毛发	是	否	否	否	是	否	哺乳类
巨蟒	冷血	鳞片	否	是	否	否	否	是	爬行类
鲑鱼	冷血	鳞片	否	是	否	是	否	否	鱼类
鲸	恒温	毛发	是	否	否	是	否	否	哺乳类
蛙	冷血	无	否	是	否	有时	是	是	两栖类
巨蜥	冷血	鳞片	否	是	否	否	是	否	爬行类
蝙蝠	恒温	毛发	是	否	是	否	是	否	哺乳类
猫	恒温	皮	是	否	否	否	是	否	哺乳类
豹纹鲨	冷血	鳞片	是	否	否	是	否	否	鱼类
海龟	冷血	鳞片	否	是	否	有时	是	否	爬行类
豪猪	恒温	刚毛	是	否	否	否	是	是	哺乳类
鳗	冷血	鳞片	否	是	否	是	否	否	鱼类
蝾螈	冷血	无	否	是	否	有时	是	是	两栖类

上例是属性有8个，每个属性又有多少离散的值可取。在决策树的每一个节点上我们可以按任一个属性的任一个值进行划分。比如最开始我们按：

1）表面覆盖为毛发和非毛发

2）表面覆盖为鳞片和非鳞片

3）体温为恒温和非恒温

等等产生当前节点的左右两个孩子。下面我们按GINI指数划分有：

GINI指数

总体内包含的类别越杂乱，GINI指数就越大（跟熵的概念很相似）。比如体温为恒温时包含哺乳类5个、鸟类2个，则：

$$GINI=1-[(\frac{5}{7})^2+(\frac{2}{7})^2]=\frac{20}{49}$$

体温为非恒温时包含爬行类3个、鱼类3个、两栖类2个,则

$$GINI=1-[(\frac{3}{8})^2+(\frac{3}{8})^2+(\frac{2}{8})^2]=\frac{42}{64}$$

所以如果按照“体温为恒温和非恒温”进行划分的话，我们得到GINI的增益（类比信息增益）：

$$GINI_Gain = \frac{7}{15} * \frac{20}{49} + \frac{8}{15} * \frac{42}{64}$$

最好的划分就是使得GINI_Gain最小的划分。

终止条件

一个节点产生左右孩子后，递归地对左右孩子进行划分即可产生分类回归树。这里的终止条件是什么？什么时候节点就可以停止分裂了？直观的情况，当节点包含的数据记录都属于同一个类别时就可以终止分裂了。这只是一个特例，更一般的情况我们计算χ2值来判断分类条件和类别的相关程度，当χ2很小时说明分类条件和类别是独立的，即按照该分类条件进行分类是没有道理的，此时节点停止分裂。注意这里的“分类条件”是指按照GINI_Gain最小原则得到的“分类条件”。

假如在构造分类回归树的第一步我们得到的“分类条件”是：体温为恒温和非恒温。此时：

	哺乳类	爬行类	鱼类	鸟类	两栖类
恒温	5	0	0	2	0
非恒温	0	3	3	0	2

三、剪枝

决策树为什么(WHY)要剪枝？原因是避免决策树过拟合(Overfitting)样本。前面的算法生成的决策树非常详细并且庞大，每个属性都被详细地加以考虑，决策树的树叶节点所覆盖的训练样本都是“纯”的。因此用这个决策树来对训练样本进行分类的话，你会发现对于训练样本而言，这个树表现完好，误差率极低且能够正确得对训练样本集中的样本进行分类。训练样本中的错误数据也会被决策树学习，成为决策树的部分，但是对于测试数据的表现就没有想象的那么好，或者极差，这就是所谓的过拟合(Overfitting)问题。Quinlan教授试验，在数据集中，过拟合的决策树的错误率比经过简化的决策树的错误率要高。

怎么剪枝

现在问题就在于，如何(HOW)在原生的过拟合决策树的基础上，生成简化版的决策树？可以通过剪枝的方法来简化过拟合的决策树。

剪枝可以分为两种：预剪枝(Pre-Pruning)和后剪枝(Post-Pruning),下面我们来详细学习下这两种方法： PrePrune：预剪枝，及早的停止树增长，方法可以参考见上面树停止增长的方法。PostPrune：后剪枝，在已生成过拟合决策树上进行剪枝，可以得到简化版的剪枝决策树。其实剪枝的准则是如何确定决策树的规模，可以参考的剪枝思路有以下几个： 1：使用训练集合(Training Set）和验证集合(Validation Set)，来评估剪枝方法在修剪结点上的效用 2：使用所有的训练集合进行训练，但是用统计测试来估计修剪特定结点是否会改善训练集合外的数据的评估性能，如使用Chi-Square（Quinlan，1986）测试来进一步扩展结点是否能改善整个

分类数据的性能，还是仅仅改善了当前训练集合数据上的性能。 3：使用明确的标准来衡量训练样例和决策树的复杂度，当编码长度最小时，停止树增长，如MDL(Minimum Description Length)准则。

1、Reduced-Error Pruning(REP,错误率降低剪枝） 该剪枝方法考虑将书上的每个节点作为修剪的候选对象，决定是否修剪这个结点有如下步骤组成： 1：删除以此结点为根的子树 2：使其成为叶子结点 3：赋予该结点关联的训练数据的最常见分类 4：当修剪后的树对于验证集合的性能不会比原来的树差时，才真正删除该结点 因为训练集合的过拟合，使得验证集合数据能够对其进行修正，反复进行上面的操作，从底向上的处理结点，删除那些能够最大限度的提高验证集合的精度的结点，直到进一步修剪有害为止(有害是指修剪会减低验证集合的精度) REP是最简单的后剪枝方法之一，不过在数据量比较少的情况下，REP方法趋于过拟合而较少使用。这是因为训练数据集合中的特性在剪枝过程中被忽略，所以在验证数据集合比训练数据集合小的多时，要注意这个问题。 尽管REP有这个缺点，不过REP仍然作为一种基准来评价其它剪枝算法的性能。它对于两阶段决策树学习方法的优点和缺点提供了了一个很好的学习思路。由于验证集合没有参与决策树的创建，所以用REP剪枝后的决策树对于测试样例的偏差要好很多，能够解决一定程度的过拟合问题。

2、Pessimistic Error Pruning(PEP，悲观剪枝） 先计算规则在它应用的训练样例上的精度，然后假定此估计精度为二项式分布，并计算它的标准差。对于给定的置信区间，采用下界估计作为规则性能的度量。这样做的结果，是对于大的数据集合，该剪枝策略能够非常接近观察精度，随着数据集合的减小，离观察精度越来越远。该剪枝方法尽管不是统计有效的，但是在实践中有效。 PEP为了提高对测试集合的预测可靠性，PEP对误差估计增加了连续性校正(Continuity Correction)。PEP方法认为，如果：

$$e'(t) \leqslant e'(T_t) + S_e(e'(T_t))$$

成立，则Tt应该被剪枝，

上式中：

$$e'(t) = \left[e(t) + \frac{1}{2}\right];$$

$$e'(T_t) = \sum e(i) + \frac{N_t}{2};$$

其中，e(t)为结点t出的误差；i为覆盖Tt的叶子结点；Nt为子树Tt的叶子树；n(t)为在结点t处的训练集合数量。PEP采用自顶向下的方式，如果某个非叶子结点符合上面的不等式，就裁剪掉该叶子结点。该算法被认为是当前决策树后剪枝算法中经度比较高的算法之一，但是饿存在有缺陷。首先，PEP算法是唯一使用Top-Down剪枝策略，这种策略会导致与先剪枝出现同样的问题，将该结点的某子节点不需要被剪枝时被剪掉；另外PEP方法会有剪枝失败的情况出现。 虽然PEP方法存在一些局限性，但是在实际应用中表现出了较高的精度,。两外PEP方

法不需要分离训练集合和验证机和，对于数据量比较少的情况比较有利。再者其剪枝策略比其它方法相比效率更高，速度更快。因为在剪枝过程中，树中的每颗子树最多需要访问一次，在最坏的情况下，它的计算时间复杂度也只和非剪枝树的非叶子节点数目成线性关系。

Cost-Complexity Pruning(CCP、代价复杂度) CCP方法包含两个步骤： 1：从原始决策树T0开始生成一个子树序列{T0、T1、T2、...、Tn},其中Ti+1是从Ti总产生，Tn为根节点 2：从子树序列中，根据树的真实误差估计选择最佳决策树。

对于分类回归树中的每一个非叶子节点计算它的表面误差率增益值α。

$$\alpha = \frac{R(t) - R(T_t)}{|N_{T_t}| - 1}$$

$|N_{T_t}|$是子树中包含的叶子节点个数;

$R(t)$是节点t的误差代价，如果该节点被剪枝;

$R(t) = r(t) * p(t)$

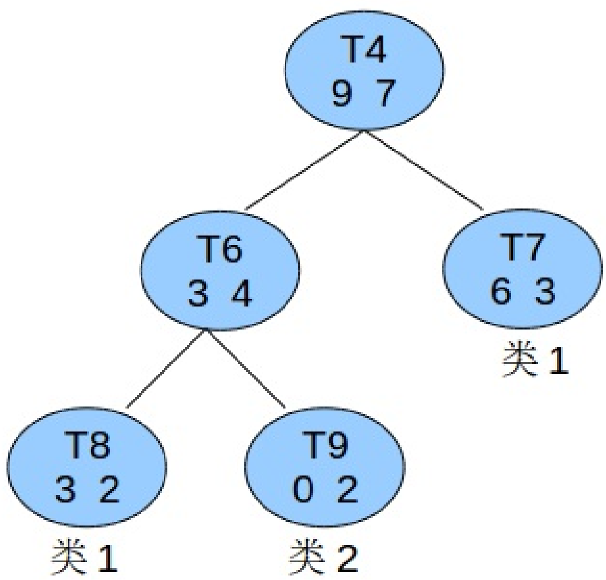
r(t)是节点t的误差率;

p(t)是节点t上的数据占所有数据的比例。

$R(T_t)$是子树T_t的误差代价，如果该节点不被剪枝。它等于子树T_t上所有叶子节点的误差代价之和。

比如有个非叶子节点t4如图所示：

比如有个非叶子节点t4如图所示：



已知所有的数据总共有60条，则节点t4的节点误差代价为：

$$R(t)=r(t)*p(t)=\frac{7}{16}*\frac{16}{60}=\frac{7}{60}$$

子树误差代价为：

$$R(T_t)=\sum R(i)=(\frac{2}{5}*\frac{5}{60})+(\frac{0}{2}*\frac{2}{60})+(\frac{3}{9}*\frac{9}{60})=\frac{5}{60}$$

以t4为根节点的子树上叶子节点有3个，最终：

$$\alpha=\frac{7/60-5/60}{3-1}=\frac{1}{6}$$

找到α值最小的非叶子节点，令其左右孩子为NULL。当多个非叶子节点的α值同时达到最小时，取$|N_{T_t}|$最大的进行剪枝。

剪枝过程特别重要，所以在最优决策树生成过程中占有重要地位。有研究表明，剪枝过程的重要性要比树生成过程更为重要，对于不同的划分标准生成的最大树(Maximum Tree)，在剪枝之后都能够保留最重要的属性划分，差别不大。反而是剪枝方法对于最优树的生成更为关键。